版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

HZ BOOKS 华章IT

ARinChina

HISCENE 亮风台信息科技

60余位AR/VR开发行业一线公司CTO审定内容且联袂推荐！

本书由ARinChina、亮风台、加速想象力等数十家AR/VR公司官方推荐

AR/VR Development Step By Step

AR与VR开发实战

张克发 赵兴 谢有龙◎等著



加速想象力必备教材

购书送价值**2800**元线上课程

AR开发重点讲解：主流的AR开发SDK如Vuforia、HiAR、Wikitude、VoidAR、EasyAR等，并结合Unity 3D引擎制作各种类型的AR产品。

VR开发重点讲解：主流的硬件如Cardboard、Gear VR、Oculus VR、HTC Vive等，进行VR应用开发的流程及制作各种类型的VR产品。

机械工业出版社
China Machine Press

THINK 加速想象力
Thinker Boost

本书为加速想象力必备教材之一，配套的加速想象力相关课程可让学习效果事半功倍、锦上添花。

加速想象力自 2015 年 9 月正式对外公开授课以来，已开设 AR/VR 基础开发、传统游戏开发、Unity 高端就业班、Unreal 高端就业班、AR/VR 专业 3D 美术班、AR/VR 暑期集训班、AR/VR 周末集训班等十余种基础、精品课程。

让有志加入 AR/VR 行业的学生或青年更好地掌握技术，企业联动生态为行业求学者提供就业平台，提升他们的综合竞争力，这便是加速想象力的使命。

目前，加速想象力已累计培育 AR/VR 优秀人才 860 余人，在培 220 余人，企业客户及项目包括腾讯、陕西科技大学、中国船舶第 705 研究所、安徽省高教系统「双师素质」增强现实培训项目等，并在北京、上海、青岛、西安、硅谷等地设有分校，实乃国内领先的 AR/VR 人才培育及实训基地。

本书行业专业顾问

张明军	CEO	ARinChina
廖春元	CEO	亮风台
徐　强	CEO	黑晶科技
袁　元	CEO	域圆科技
翁孝清	CEO	微景文化
祖厚超	CEO	青瞳视觉
吴四勇	总经理	知问教育
张克发	CEO	加速想象力
唐荣兴	联合创始人 &COO	亮风台
朱　靖	产品总监	联想创投集团
余　海	CEO	Realmax
俞志杰	产品总监 CEO	璟世科技
罗翔宇	CEO& 首席制作人	铃空游戏
史晓刚	CEO	枭龙科技
陈德辉	联合创始人	我的天科技

本书行业专业顾问

崔海庆	合伙人	暴风魔镜
原旭晗	CEO	旗木堂
胡金鑫	联合创始人	影创科技
熊剑明	CEO	小熊尼奥
孙　谦	CEO	幻实科技
赵　亮	CTO	璟世科技
张天夫	CEO	微力互动
李蓓贝	CEO	它说
谈　飞	研发总监	Realmax
古　鉴	交互技术负责人	暴风魔镜
周粤臻	CEO	童萌汇
王亚楠	CEO	热玩游戏
邱　剑	CEO	无忧我房
潘博航	创始人	VRplay
吴泽顺	CEO	思酷网络
何文艺	CEO	乐客 VR
刘鸿惠	联合创始人	幻眼科技
秦　凯	CEO	BeanVR



AR/VR Development
Step By Step

AR与VR
开发实战

张克发 赵兴 谢有龙◎等著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

AR与VR开发实战 / 张克发等著. —北京：机械工业出版社，2016.11（2017.7重印）

ISBN 978-7-111-55330-4

I. A… II. 张… III. 计算机仿真 IV. TP391.9

中国版本图书馆CIP数据核字（2016）第258617号

AR与VR开发实战

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）

责任编辑：曲 熠　　责任校对：殷 虹

印　　刷：北京瑞德印刷有限公司　　版　　次：2017年7月第1版第3次印刷

开　　本：185mm×260mm 1/16　　印　　张：17.5

书　　号：ISBN 978-7-111-55330-4　　定　　价：69.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88379426 88361066　　投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259　　读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光/邹晓东

前言

增强现实（Augmented Reality）简称 AR，是一种实时计算摄像机捕捉到的现实影像的位置及角度并加上相应虚拟信息的技术。这种技术不仅能展现真实世界的信息，还能通过增加虚拟信息而对现实世界进行增强，所以称为增强现实。AR 可应用于军事、医疗、工业、教育等众多领域，随着硬件性能的提升和软件解决方案的日趋成熟，AR 产品将越来越多地走入人们的视线。

虚拟现实（Virtual Reality）简称 VR，它通过计算机图形系统以及多种传感器控制设备将人带入虚拟世界中，并与虚拟世界产生各种交互，具有强烈的沉浸式体验。VR 将在游戏、影视、城市规划、室内设计、地产、教育等多个领域为人们提供更加真实、震撼的全新体验。

随着 AR 和 VR 技术的飞速发展，由此产生的众多应用场景将需要大量技术人才，尤其是程序设计人才。本书的目的是为程序开发者提供全面、权威的参考资料，并附有大量真实案例，详细讲解 AR、VR 开发的流程和步骤。

本书分为两大部分，第一部分是“AR 开发”，共 12 章：

- 第 1 章简要介绍 AR 技术的概念和常用的 AR 解决方案。
- 第 2 章主要介绍基于 Vuforia SDK 的应用开发。
- 第 3 ～ 9 章主要介绍 AR 项目中常见的案例以及实现方法，例如透明视频、AR 对战游戏、涂色类产品等。
- 第 10 章重点讲解国内 AR 解决方案 HiAR SDK 的开发。
- 第 11 章简要介绍其他 AR SDK 的使用。
- 第 12 章简要介绍 AR 创建工具“找趣”的使用。

第二部分是“VR 开发”，共 6 章：

- 第 13 章简要介绍 VR 技术的概念和主流的 VR 设备。
- 第 14 章主要介绍基于 Google Cardboard 开发 VR 应用。
- 第 15 章主要介绍 HTC Vive 的常用功能及其实现。
- 第 16、17 章通过案例方式讲解基于 HTC Vive 的项目开发。
- 第 18 章简要介绍目前市场上的其他 VR 设备及相关技术。

本书专注于 AR 和 VR 开发，因此不过多涉及编程语言的基础知识及引擎的使用方法，阅读本书需要先掌握 C# 编程技术，并能够熟练使用 Unity 3D 引擎。如果你没有 C# 和 Unity 的使用经验，那么在阅读本书之前需要先学习相关技术。

在编写本书的过程中，我们得到了亮风台科技、克科技、RealMax 等企业的技术支持和宝贵意见，在此表示衷心的感谢。感谢 ARinChina CEO 张明军提供的优质平台与宝贵资源，感谢 ARinChina 同事董艳超、张荣为本书的出版辛苦奔波，感谢 ARinChina 站长赵亮、ARinChina 技术论坛版主赵忠立提供高质量的技术支持，感谢付旭耀提供美术支持。最后，特别感谢 ARinChina 工程师赵兴、谢有龙、陈家豪、常壮等同事，他们在本书编著过程中付出了无数汗水。

本书是非常好的 AR 和 VR 入门教程，适用于高校计算机相关专业学生学习 AR 和 VR 技术，同时由于书中附有大量案例，所以同样适用于企业开发人员。如果你在项目中遇到相似的需求，或许也可以从本书中获取一些灵感和帮助。

如果本书能为你的学习或工作带来帮助和提升，将是我们莫大的荣幸。真诚希望本书的读者给我们更多的反馈和意见，帮助我们取得更大的进步和成长。在阅读本书的过程中若有疑问，欢迎加入本书 QQ 群进行交流，我们会在群里提供本书的所有资源和相关工具，QQ 群号为 588206982。

作者

2016 年 9 月

作者简介

张克发：毕业于烟台大学软件工程学院，曾在杭州、上海等地参与多款大型游戏的开发，擅长 Android 开发、Unity 3D、Node.js 等技术，有丰富的 AR、VR 开发经验，曾主导开发多款 AR、VR 产品。目前担任 ARinChina（增强现实中国）教育事业部总监，以及北京加速想象力教育咨询有限公司 CTO。

赵亮：ARinChina 站长，德国 Metaio 驻中国技术支持，奥地利 Wikitude 驻中国技术支持，目前担任上海璟世数字科技有限公司 CTO。

赵忠立：曾在西安、上海等地从事三年 AR 开发，目前担任上海璟世数字科技有限公司技术总监，擅长 Unity 3D 与 Android 开发。

赵兴：毕业于西安邮电大学数学系，擅长 Unity 3D 开发与算法研究，目前在 ARinChina 担任 AR、VR 开发工程师。

谢有龙：在 ARinChina 担任 AR、VR 开发工程师，擅长 HTC Vive、Hololens 等 AR、VR 技术。

陈家豪：毕业于西北大学软件工程学院，擅长 3D 美术制作、Unity 3D 开发、Unreal Engine 开发。目前在 ARinChina 担任 3D 设计师及 AR、VR 开发工程师。

常壮：毕业于西北工业大学，在 ARinChina 担任 AR、VR 开发工程师。

目录

前　言
作者简介

第一部分　AR 开发

第 1 章　AR 技术简介 / 2

第2 章　基于 Vuforia SDK 开发 AR 应用 / 4

2.1　准备 Vuforia 开发环境 / 4
2.2　创建 Vuforia 案例 / 7
2.3　创建 AR 视频 / 14
2.4　文字识别 / 15
2.5　柱形识别 / 18
2.6　立方体识别 / 21
2.7　3D 物体识别 / 24
2.8　云识别 / 26
2.9　智能地形 / 29
2.10　虚拟按钮 / 33
2.11　帧标记识别 / 37
2.12　自定义目标识别 / 40

第3 章　AR 内容交互 / 46

3.1　动态加载 AR 模型 / 46
3.2　模型脱卡功能的实现 / 50
3.3　手势控制 / 53

第 4 章　AR 互动大屏：透明视频的实现 / 57

4.1　使用 AE 制作带 Alpha 通道的视频 / 57
4.2　编写可播放透明视频的 Shader / 59
4.3　开启摄像头并实现互动大屏的视频播放 / 64

第5章 AR动画：AR与Flash动画的结合 / 68

5.1 准备GAF插件 / 68
5.2 实现动画播放 / 69
5.3 用AR方式实现扫描 / 70

第6章 AR游戏开发：卡牌对战 / 71

6.1 多图识别生成模型 / 71
6.2 卡牌对战功能的实现 / 74

第7章 涂色类AR开发 / 87

7.1 UV的概念 / 87
7.2 使用Photoshop绘制涂色的识别图 / 88
7.3 使用3D Max处理模型的UV / 89
7.4 搭建Vuforia AR开发环境 / 94
7.5 核心功能之通过C#脚本获得变量 / 96
7.6 核心功能之通过C#脚本向Shader传递变量 / 101
7.7 在Unity中完善项目 / 104

第8章 使用陀螺仪模拟SLAM功能 / 106

8.1 SLAM的概念 / 106
8.2 模拟SLAM功能的实现 / 106

第9章 Unity与原生代码之间的交互 / 110

9.1 Unity与Android原生代码之间的交互 / 110
9.2 Unity与iOS原生代码之间的交互 / 115

第10章 使用HiAR SDK开发AR应用 / 123

10.1 搭建开发环境 / 123
10.2 导入SDK / 124
10.3 创建HelloWorld / 126
10.4 导出Android工程 / 130
10.5 导出iOS工程 / 135
10.6 使用本地识别包 / 137
10.7 开发云识别应用 / 140
10.8 制作云识别内容 / 143
10.9 创建视频AR应用 / 147
10.10 使用HiAR创建涂色类AR应用 / 149
10.11 多图识别 / 152
10.12 动态加载 / 154
10.13 设置场景中心点 / 156

第 11 章　其他 AR SDK 简介 / 159

11.1　基于 Wikitude SDK 开发 AR 应用 / 159

11.2　基于 VoidAR SDK 开发 AR 应用 / 163

第 12 章　AR 创建工具：找趣 (realcast) 浏览器 / 178

12.1　什么是找趣 / 178

12.2　找趣后台 / 178

12.3　创建场景 / 181

12.4　制作场景 / 183

12.5　如何使用找趣 / 187

12.6　管理场景 / 项目 / 187

12.7　事件交互的定义 / 189

12.8　模型动画的播放 / 191

第二部分　VR 开发

第 13 章　VR 技术简介 / 194

第 14 章　基于 Google Cardboard 开发 VR 应用 / 197

14.1　Cardboard SDK for Unity 环境配置 / 197

14.2　利用 Cardboard 创建 VR 场景 / 200

14.3　Cardboard 中 GUI 的设置 / 201

14.4　Cardboard 中 3D 物体的选取 / 203

14.5　Cardboard 中凝视触发操作的实现 / 204

14.6　Cardboard 中全景相册的实现 / 209

14.7　Cardboard 中全景视频播放的实现 / 211

第 15 章　基于 HTC Vive 开发 VR 应用 / 214

15.1　Vive 设备的安装 / 214

15.2　搭建 Vive 开发环境 / 217

15.3　创建 Vive 游戏场景 / 218

15.4　Vive 中的手柄交互 / 220

15.5　功能开发之扳机键 (Trigger) / 225

15.6 功能开发之触摸板（Touchpad） / 228
15.7 功能开发之功能键（ApplicationMenu） / 230
15.8 功能开发之手柄振动（TriggerHapticPulse） / 230
15.9 功能开发之瞬移（Teleporter） / 231
15.10 功能开发之左右侧键（Grip） / 236
15.11 功能开发之 The Lab Renderer / 237

第 16 章 基于 HTC Vive 实现房间漫游 / 239

16.1 准备工作 / 239
16.2 基本平台搭建 / 239
16.3 导入房间模型 / 240
16.4 基于 Vive 手柄的场景移动 / 241
16.5 基于 Vive 手柄的场景旋转 / 244
16.6 手柄与场景物体的交互 / 245

第 17 章 基于 HTC Vive 开发投篮游戏 / 250

17.1 准备素材 / 250
17.2 场景搭建 / 250
17.3 导出与运行 / 264

第 18 章 其他 VR 设备简介 / 266

18.1 Oculus VR 简介 / 266
18.2 GearVR 简介 / 267
18.3 大朋 VR 简介 / 268
18.4 PSVR 简介 / 269

第一部分

AR 开发

- 第 1 章　AR 技术简介
- 第 2 章　基于 Vuforia SDK 开发 AR 应用
- 第 3 章　AR 内容交互
- 第 4 章　AR 互动大屏：透明视频的实现
- 第 5 章　AR 动画：AR 与 Flash 动画的结合
- 第 6 章　AR 游戏开发：卡牌对战
- 第 7 章　涂色类 AR 开发
- 第 8 章　使用陀螺仪模拟 SLAM 功能
- 第 9 章　Unity 与原生代码之间的交互
- 第 10 章　使用 HiAR SDK 开发 AR 应用
- 第 11 章　其他 AR SDK 简介
- 第 12 章　AR 创建工具：找趣（realcast）浏览器

第1章 AR技术简介

增强现实（Augmented Reality，AR）是一种实时计算摄影机影像的位置及角度并加上相应图像的技术，这种技术的目标是在屏幕上把虚拟世界套在现实世界中并进行互动。这种技术于20世纪90年代提出，随着随身电子产品运算能力的提升，其用途将会越来越广。

目前对于增强现实有两种通用的定义。一种定义是北卡大学罗纳德·阿祖玛（Ronald Azuma）于1997年提出的，他认为增强现实包括三个方面的内容：将虚拟物与现实结合、即时互动、三维。另一种定义是保罗·米尔格拉姆（Paul Milgram）和岸野文郎（Fumio Kishino）于1994年提出的现实－虚拟连续统（Milgram's Reality-Virtuality Continuum）。后者将真实环境和虚拟环境分别作为连续统的两端，它们的中间地带称为“混合现实”（Mixed Reality），其中靠近真实环境的是增强现实（Augmented Reality），靠近虚拟环境的则是虚拟现实（Virtual Reality）。

增强现实技术在工业上的应用已有不少，目前主要用于大型机械的维修和制造。另外，增强现实技术在教育、医疗、房产、游戏等多个领域的应用同样很多。

总的来讲，增强现实技术现在还处于起步阶段，大多数AR应用都运行在智能手机上，通过图像识别等技术实现简单的增强现实效果。其采用的解决方案大部分是利用Unity 3D等游戏引擎配合AR SDK进行开发，目前最受欢迎的SDK有国外的Vuforia SDK以及国内的HiAR SDK等。

Vuforia SDK最早是高通（Qualcomm）公司的AR解决方案，后被PTC收购，目前仍致力于AR以及MR技术的研究。Vuforia集成了图像识别和物体识别，支持与三星Gear VR或微软Hololens相结合的应用开发。

HiAR是亮风台研发的AR基础开发平台，提供HiAR SDK、HiAR云、管理后台和浏览器幻镜等一系列完整的AR开发支持，专注于为开发者带来最好的AR服务。HiAR SDK功能全面，支持平面识别、3D识别、空间建模、人脸增强和姿态跟踪。2016年里约奥运会，在HiAR SDK的支持下，腾讯手机QQ于世界范围内展开AR火炬传递活动，创造了AR互动破亿次的吉尼斯世界纪录。HiAR云平台是国内首款AR云，支持亿级图库秒级搜索，识

别准确率高达96%，是国内AR平台中准确率最高、响应速度最快、识别图库最大的AR云。迄今为止，HiAR已成为全球用户量最多的AR开发平台之一，覆盖用户超过8亿，三星、腾讯、百度、华为、OPPO、大疆、搜狐等企业均在HiAR获取AR技术支持与服务。

本书将介绍图形图像识别、简单的几何体识别、3D物体识别等目前常用的AR开发技术，并结合Vuforia、HiAR、Wikitude、VoidAR等知名AR开发工具包（SDK），以案例形式讲解AR在各个领域中的应用方案。

由于本书将重点讲解AR技术开发，因此假设读者已经掌握了C#编程语言以及Unity 3D开发，如果读者对此没有接触，建议首先学习C#以及Unity 3D。

第2章 基于 Vuforia SDK 开发 AR 应用

2.1 准备 Vuforia 开发环境

1. Vuforia 简介

Vuforia 是一款能为现实世界物体带来互动体验的 AR 开发平台，旨在帮助开发者打造全新级别的真实世界物品与虚拟物品的互动。它使用计算机视觉技术来实时地识别和跟踪平面图像以及简单的 3D 物体，使开发者能够在现实世界和数字体验之间架起桥梁。

Vuforia 通过 Unity 游戏引擎扩展提供了 C、Java、Objective-C 和 .NET 语言的应用程序编程接口，能够同时支持 iOS 和 Android 的原生开发，使得开发者在 Untiy 引擎中开发的 AR 应用很容易移植到 iOS 和 Android 平台上。

2. 注册成为 Vuforia 用户

（1）打开 Vuforia 官网 https://developer.vuforia.com/。

（2）如果你已经是 Vuforia 注册用户，就可以直接输入邮箱和密码登录 Vuforia 管理后台；如果你还没有注册过 Vuforia 用户，则需要点击注册并填写相关注册信息。

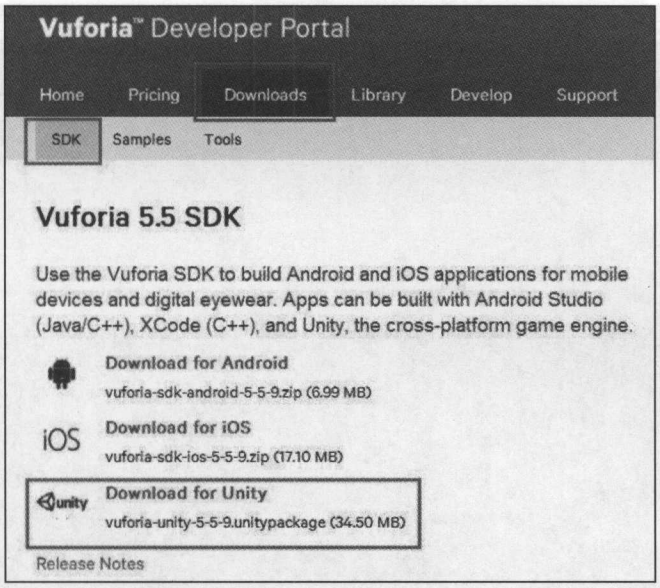
（3）在完成注册信息填写后，Vuforia 会给注册邮箱发送一封激活邮件，登录邮箱按照提示操作即可激活 Vuforia 账户。

（4）在激活 Vuforia 账户后就可以点击登录进入 Vuforia 管理后台了，如下图所示。



3. 下载 Vuforia SDK for Unity

（1）点击上图中的 Downloads 链接并按照下图所示完成操作，注意在下载列表中选择 Download for Unity。



（2）本节下载的 Vuforia SDK 版本为 vuforia-unity-5-0-5。下载解压完成后会得到后缀名为 .unitypackage 的 Unity 插件包。

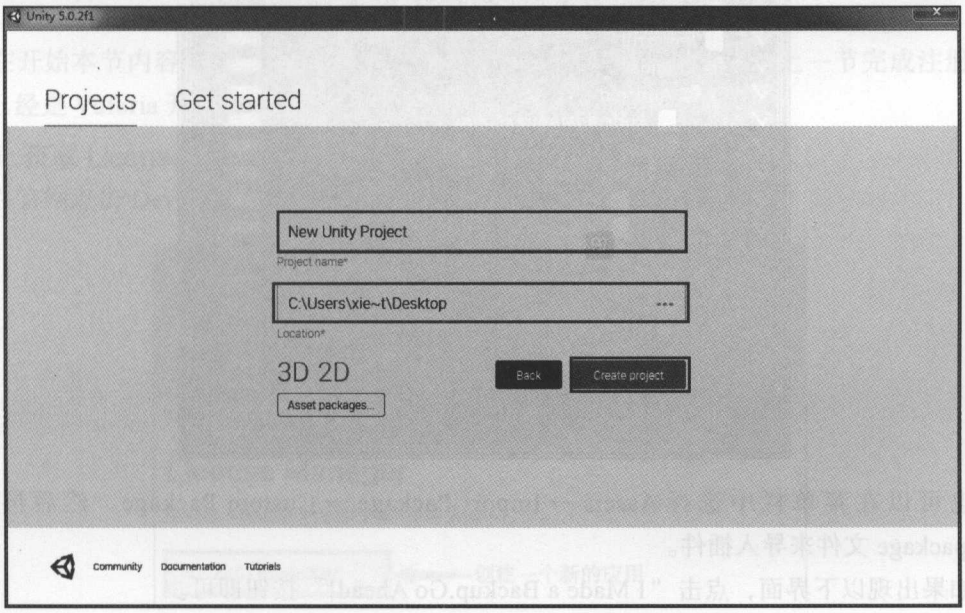
4. Vuforia 5.0.5 SDK 支持的环境

（1）Android 平台支持的最低版本是 Android 4.0.3。

（2）Unity 支持版本为 4.6.7—5.1.3p1（本节使用的 Unity 版本为 5.0.2f1）。

5. 新建 Unity 工程

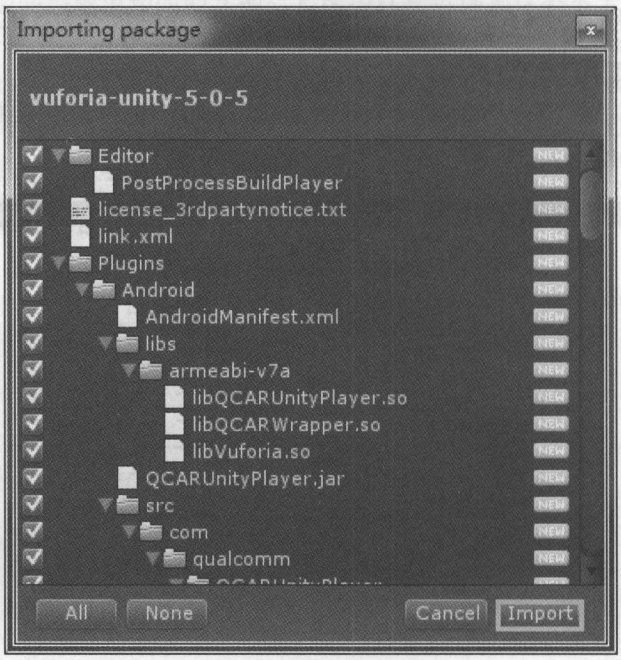
运行 Unity 程序，按照步骤创建一个新的 Unity 工程，也可以直接打开已有的 Unity 工程来进行 Vuforia SDK 导入。





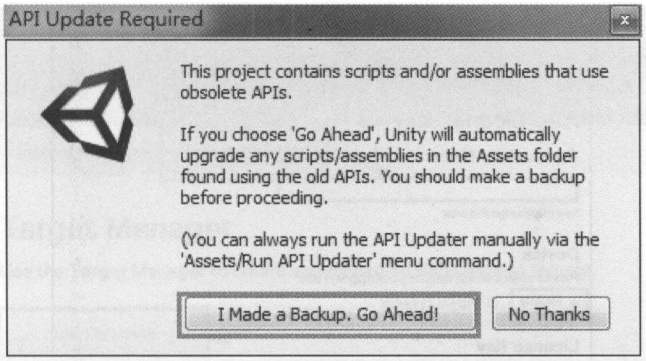
6. 导入 Vuforia SDK for Unity 插件

双击在步骤 3 中下载的 .unitypackage 插件包，弹出以下窗口后点击 Import。

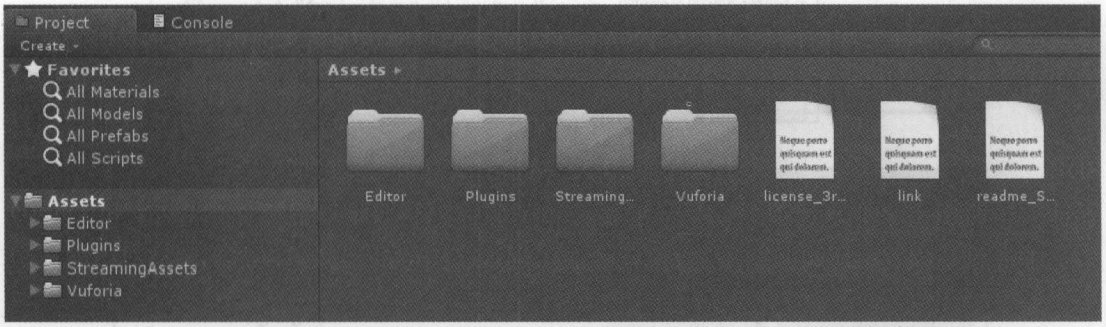


也可以在菜单栏中选择 Assets → Import Package → Custom Package，然后再选择 .unitypackage 文件来导入插件。

如果出现以下界面，点击“I Made a Backup.Go Ahead!”按钮即可。



在导入完成后，Project 窗口中显示如下：



至此，Vuforia SDK 已经成功导入了新建的 Unity 项目中，接下来可以创建开发 Demo。

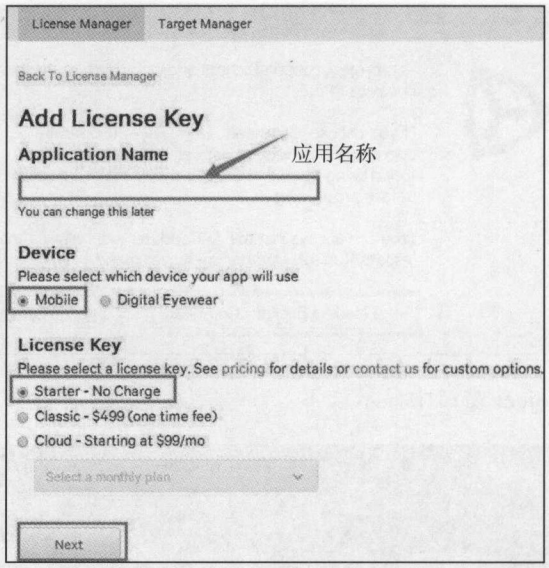
2.2 创建 Vuforia 案例

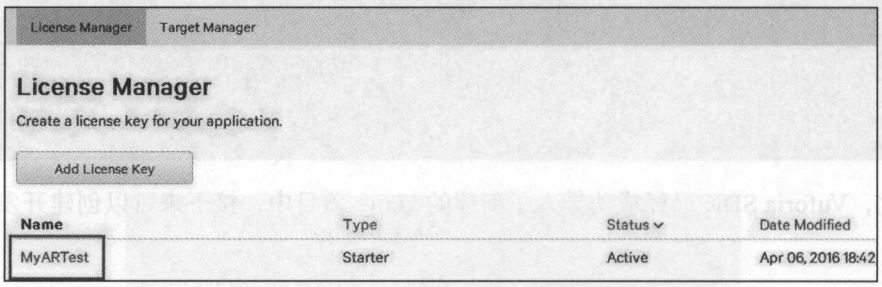
在开始本节内容之前，如果你还没有创建 Vuforia 账号，请先参照上一节完成注册，如果你已经是 Vuforia 开发者，请继续阅读接下来的内容。

1. 获取 License Key

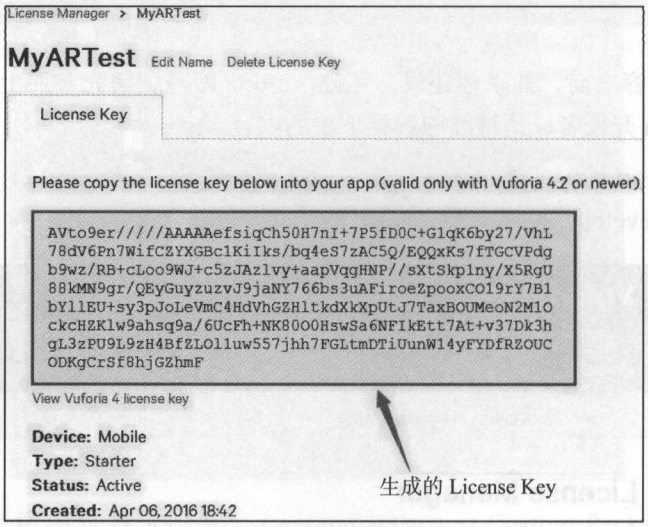
在官网点击 Develop，按照下图完成操作：







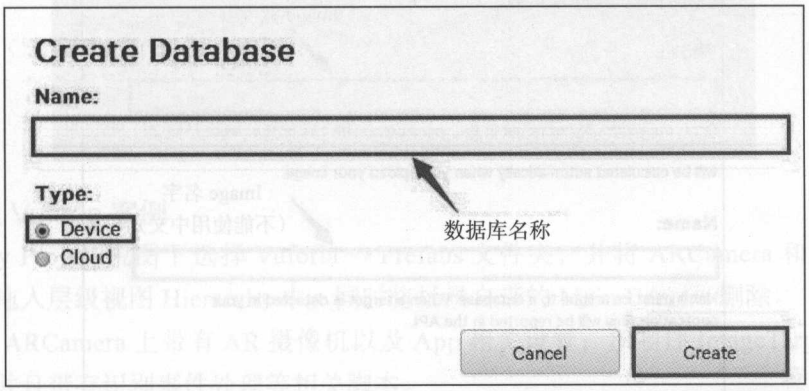
完成以上操作后，在 License Manager 中点击自己创建的应用名称，进入下面的界面：



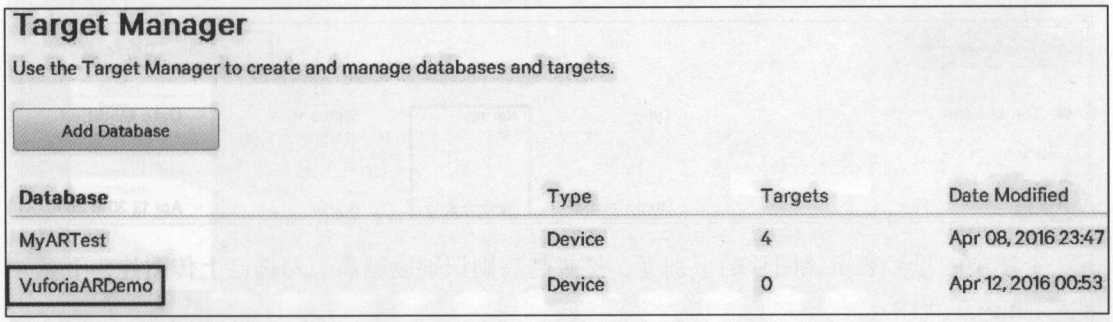
2. 导出新建应用的数据包

根据下图先创建一个新的数据库：

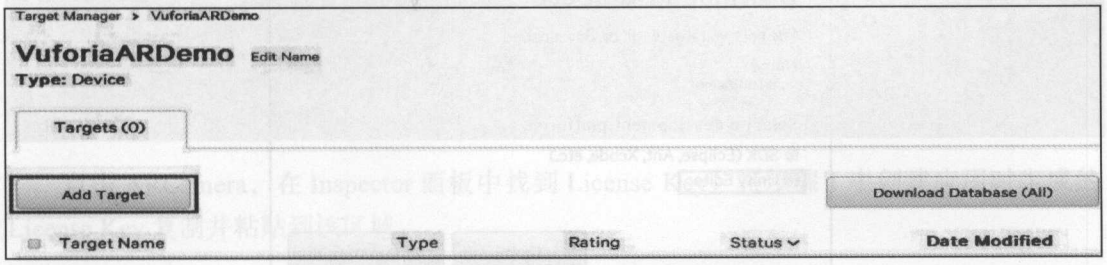




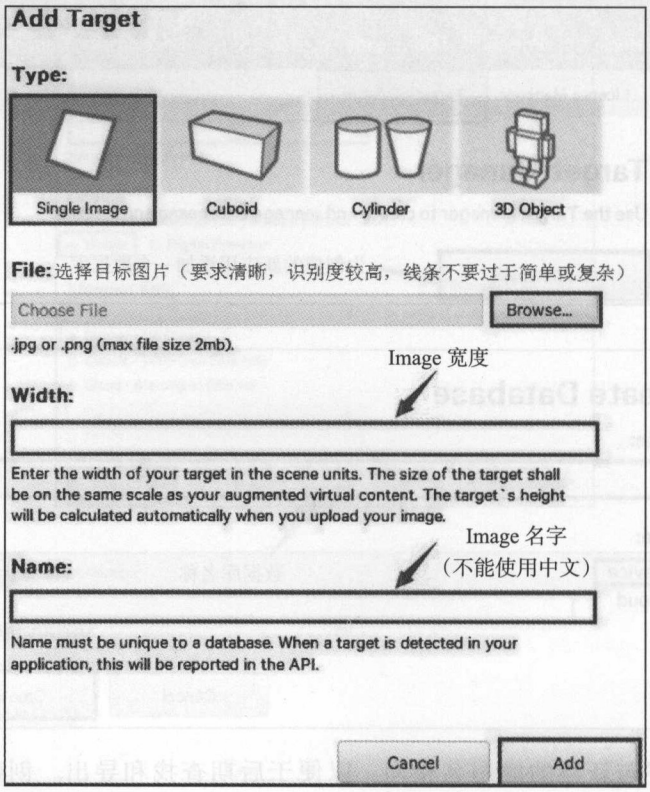
数据包名最好与新建的应用名相同，以便于后期查找和导出。创建好之后进入以下界面：



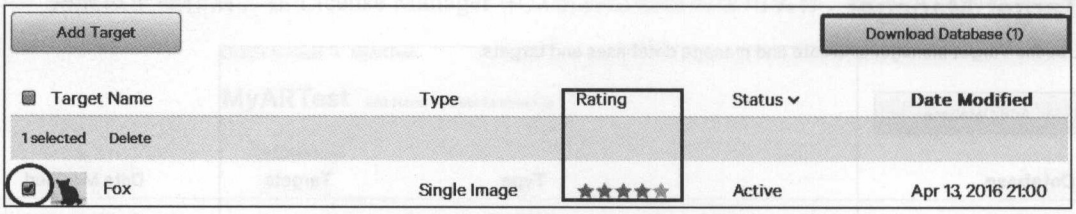
打开新建的数据包，添加识别目标：



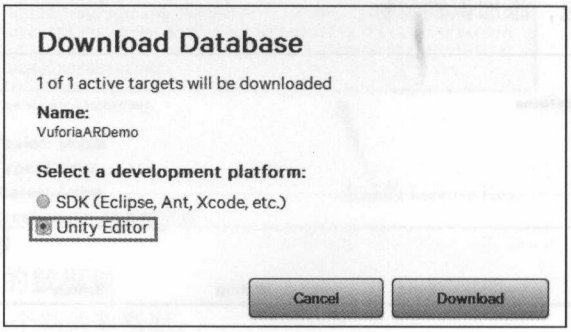
本书选择 Single Image（单个图像）作为识别图：



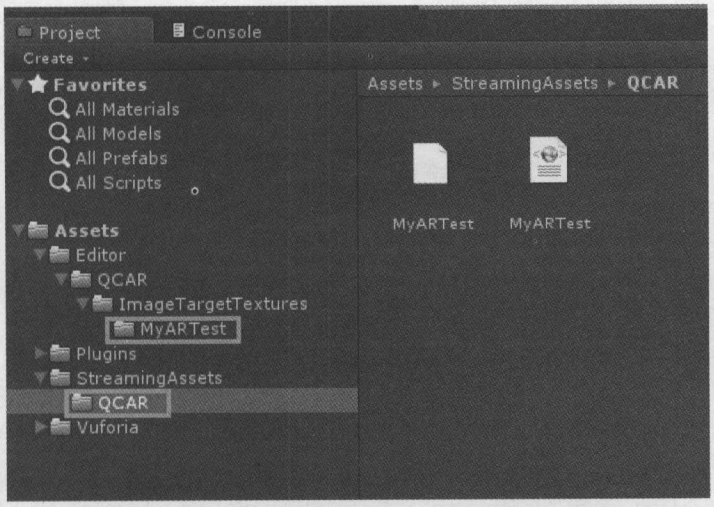
添加成功后会显示如下界面：



上图中的星号表示该图片的识别度，星级越高则识别度越高。勾选已上传图片，下载并导入 Unity 工程中。



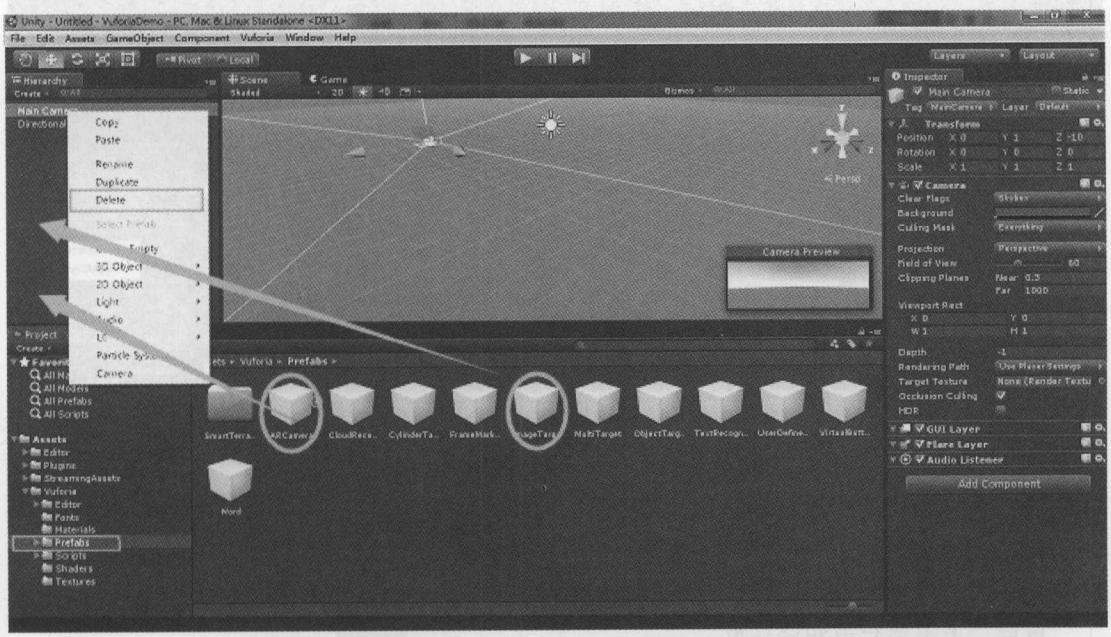
导入成功后如下图所示：



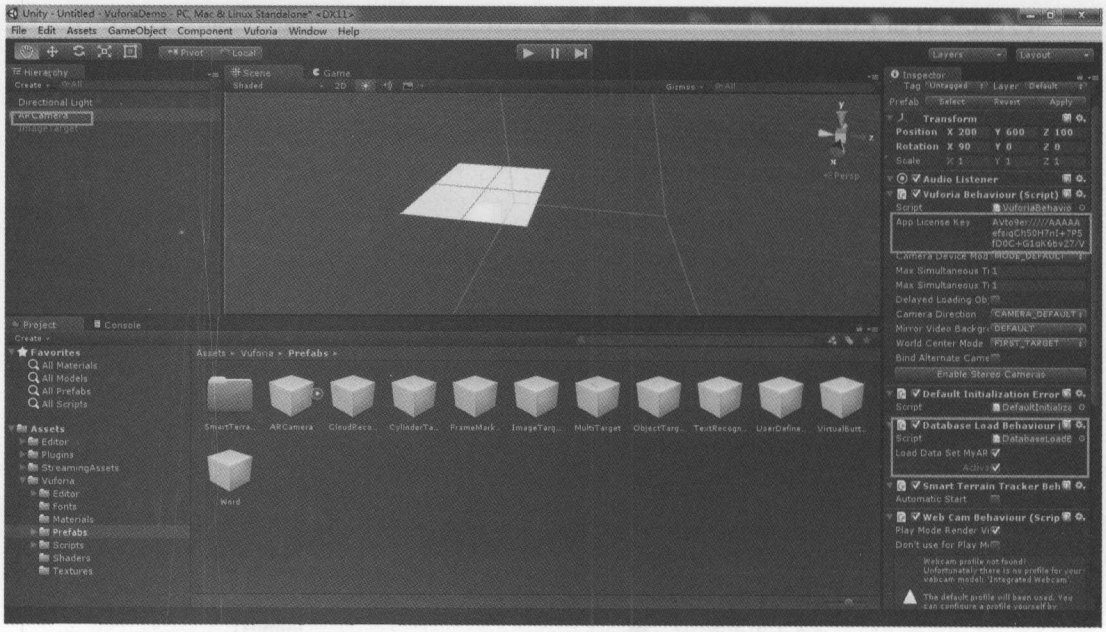
3. 创建 Vuforia 案例

在 Unity Project 视图下选择 Vuforia → Prefabs 文件夹，并将 ARCamera 和 ImageTarget 两个预制件拖入层级视图 Hierarchy 中，同时将场景自带的 Main Camera 删除。

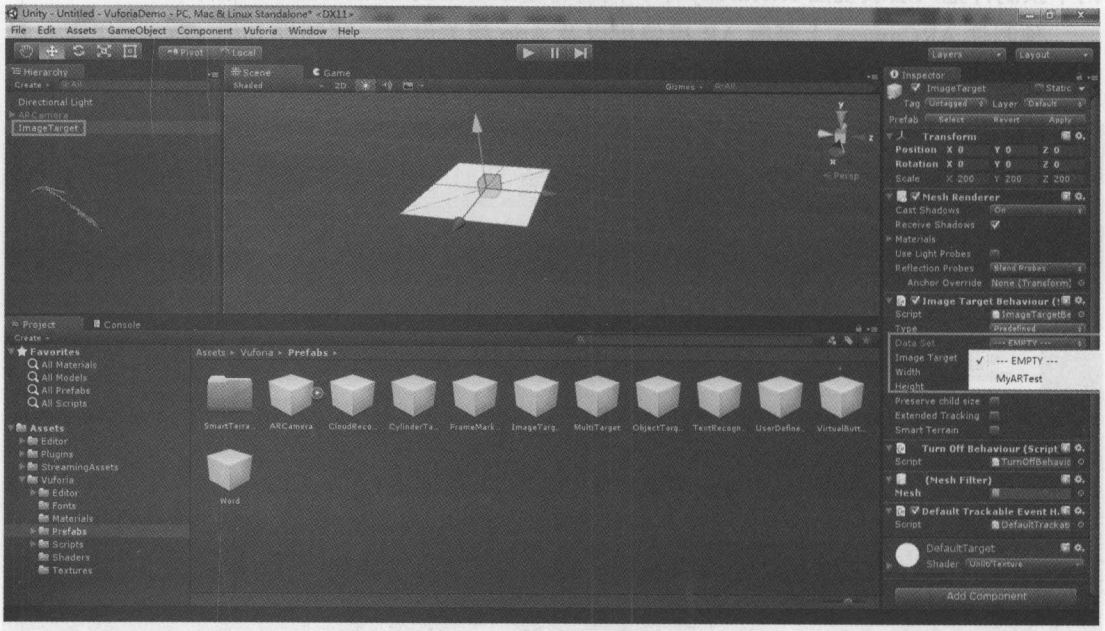
预制件 ARCamera 上带有 AR 摄像机以及 App 相关设置；预制件 ImageTarget 代表一张识别图像，并且带有识别事件处理等相关脚本。



点击 ARCamera，在 Inspector 面板中找到 License Key，将步骤 1 中创建应用时生成的 License Key 复制并粘贴到该区域。



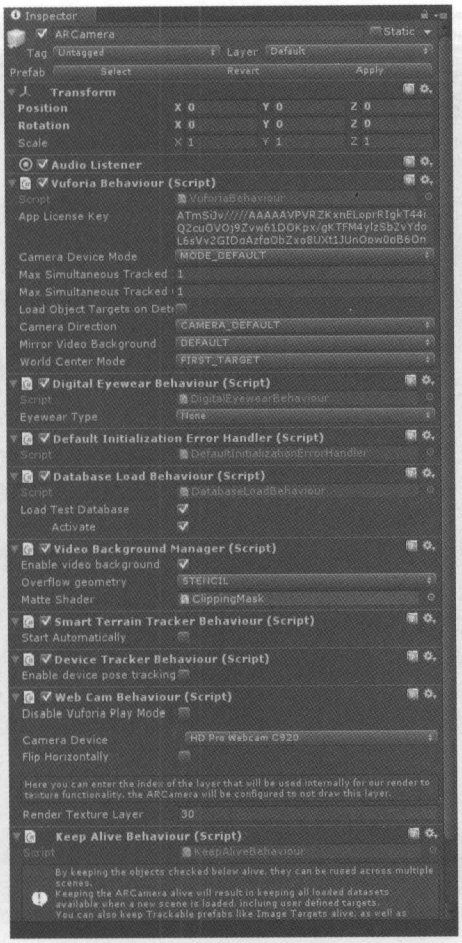
ARCamera 设置完成后点击 ImageTarget，按照下图进行设置：

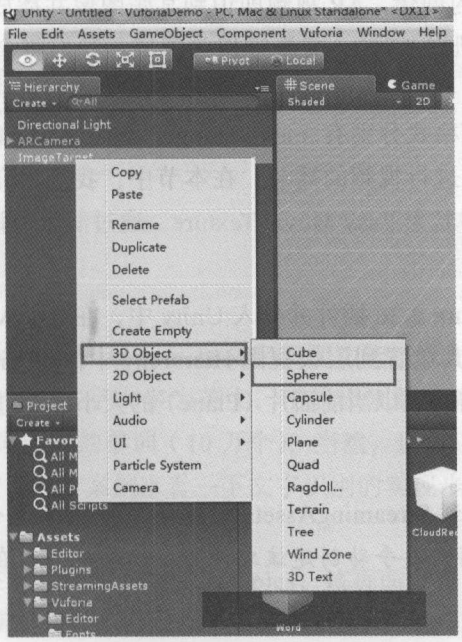


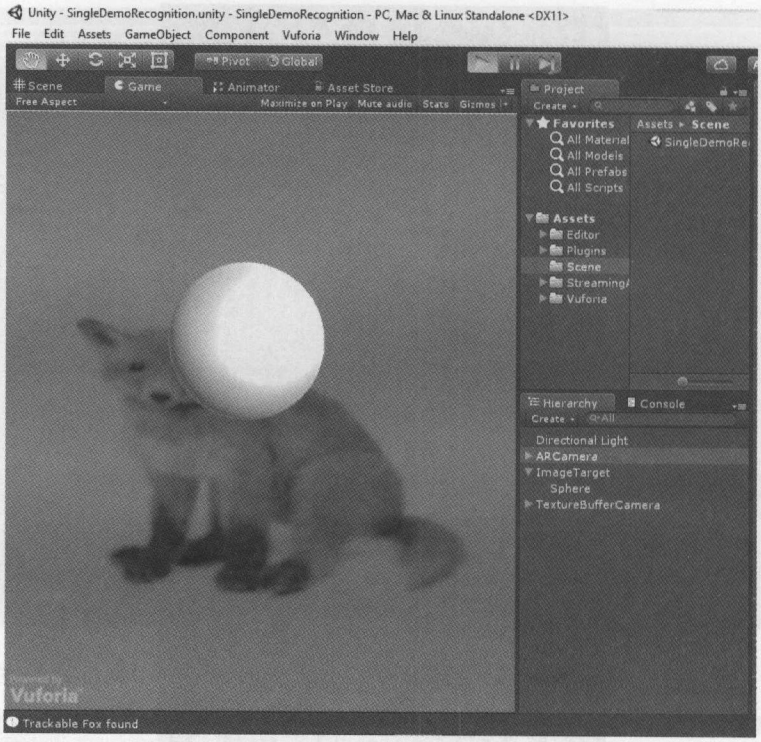
右击 ImageTarget 创建一个三维物体模型，调整三维物体与识别图像的位置关系以使其外于合适位置。

最后，点击运行来测试效果。

至此，一个简易的 Vuforia 案例创建成功。


Inspector
ARCamera
Static
Tag Untagged
Layer Default
Prefab Select Revert Apply
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Audio Listener
Vuforia Behaviour (Script)
Script VuforiaBehaviour
App License Key
ATmSdv////AAAAAVPVRZKxnELoprRIgkT44i
QZcuOVOj9Zvw61DOKpx/gkTFM4ylzSb2vYdo
L6sVv2GIDoAzfoObZxo8UXt1JUnObw0oB6On
Camera Device Mode MODE_DEFAULT
Max Simultaneous Tracked 1
Max Simultaneous Tracked 1
Load Object Targets on Deb
Camera Direction CAMERA_DEFAULT
Mirror Video Background DEFAULT
World Center Mode FIRST_TARGET
Digital Eyewear Behaviour (Script)
Script DigitalEyewearBehaviour
Eyewear Type None
Default Initialization Error Handler (Script)
Script DefaultInitializationErrorHandler
Database Load Behaviour (Script)
Script DatabaseLoadBehaviour
Load Test Database
Activate
Video Background Manager (Script)
Enable video background
Overflow geometry STENCIL
Matte Shader ClippingMask
Smart Terrain Tracker Behaviour (Script)
Start Automatically
Device Tracker Behaviour (Script)
Enable device pose tracking
Web Cam Behaviour (Script)
Disable Vuforia Play Mode
Camera Device HD Pro Webcam C920
Flip Horizontally
Here you can enter the index of the layer that will be used internally for our render to texture functionality. the ARCamera will be configured to not draw this layer.
Render Texture Layer 30
Keep Alive Behaviour (Script)
Script KeepAliveBehaviour
By keeping the objects checked below alive, they can be ruse across multiple scenes.
Keeping the ARCamera alive will result in keeping all loaded datasets available when a new scene is loaded, incluing user defined targets.
You can also keep Trackable prefabs like Image Targets alive, as well as


Unity - Untitled - VuforiaDemo - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Vuforia Window Help
Pivot Local
Hierarchy
Create
Scene
Shaded
Game
2D
Directional Light
ARCamera
ImageTarget
Copy
Paste
Rename
Duplicate
Delete
Select Prefab
Create Empty
3D Object
2D Object
Light
Audio
UI
Particle System
Camera
Cube
Sphere
Capsule
Cylinder
Plane
Quad
Ragdoll...
Terrain
Tree
Wind Zone
3D Text
Project
Create
Assets
Editor
Plugins
StreamingAssets
Vuforia
Editor
Fonts
Word



2.3 创建 AR 视频

本节将介绍 AR 视频的创建，AR 视频即识别某张图像并播放与所识别图像对应的视频文件，一般用于企业宣传册、广告、书籍等相关领域。我们将会继续使用 Unity 和 Vuforia SDK 来开发 AR 视频应用，关于 Vuforia SDK 的相关配置可参阅前面的章节。

Unity 默认支持的视频格式分别有 .mov、.mpg、.mpeg、.mp4、.avi 和 .asf，我们在项目应用中一般采用 .mp4 文件进行视频的播放。在本节中，我们将介绍一款用于移动手机视频播放的 Unity 插件，该插件名为 Easy Movie Texture，通过简单的设置即可完成视频的播放。

1. 插件导入

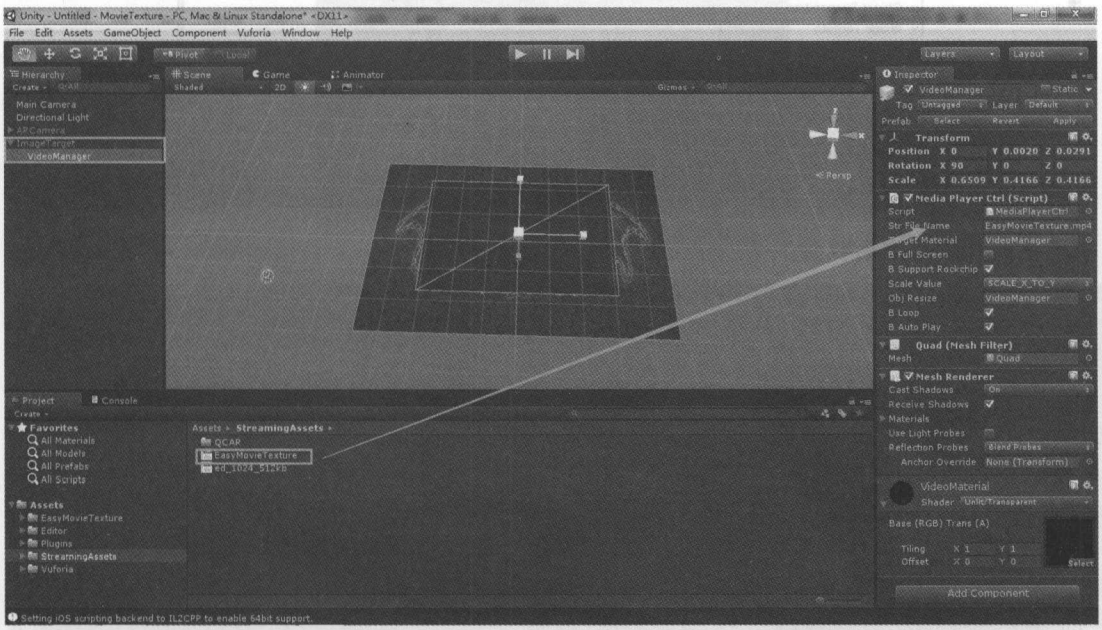
下载 Easy Movie Texture 2.36 插件并导入 Unity 中，在 EasyMovieTexture 文件夹下找到 VideoManager 预制件并将其放置到层级视图 Hierarchy 中，将 VideoManager 作为识别图像的子物体，该预制件可以将视频以当前面片（Plane）的大小比例进行播放。

注意

在 Assets 文件夹中找到 StreamingAssets 文件夹，将视频文件放在该文件夹下。如果 Assets 中没有该文件夹，则需要手动创建该文件夹，如果视频放在其他文件夹下，将无法加载视频。

2. 设置视频播放选项

选中 VideoManager，在 Inspector 面板的 Str File Name 选项中填入视频名称，如 a.mp4。该组件还可以设置是否自动播放（B Auto Play）、是否循环播放（B Loop）等属性。该预制件本质上是一个面片和控制播放的 Media Player Ctrl 组件，所以可以手动调整面片相对于识别图的大小和位置。



3. 注意事项

（1）Easy Movie Texture 插件不支持 PC 端的视频播放，只支持 Android 操作系统和 iOS 平台。

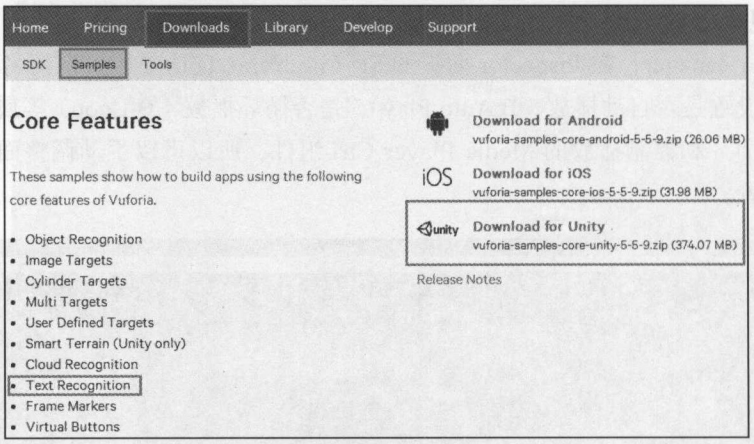
（2）如果需要从网络加载视频文件，则只需将视频文件的网络 URL 添加到 Str File Name 属性上，无需其他操作。

2.4 文字识别

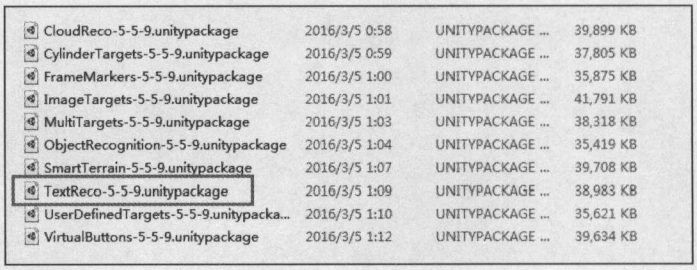
本节介绍如何使用 Vuforia 和 Unity 实现文字识别功能。Vuforia 对文字识别有一定的要求，首先，Vuforia 能够识别的文字大多为英文，不能识别数字和中文。其次，默认能够识别的英文单词必须是官方词库中的单词（10 万个），当然，还可以自定义添加英文单词。接下来我们通过 Vuforia 官方自带的案例来看一下文字识别的实现步骤。

1. 插件导入

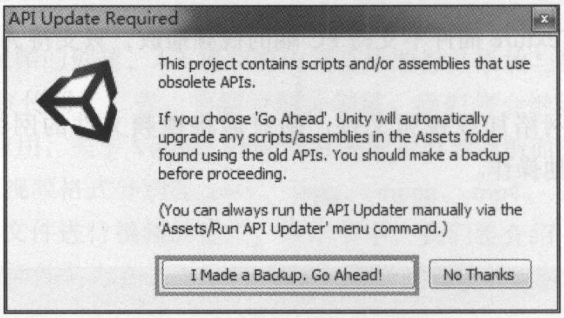
登录 Vuforia 官网，在下载页面中找到 Samples，该页面中有 Vuforia SDK 提供的基本功能对应的案例，下载下图所示文件。



下载并解压之后找到下图所示 Unity 包并导入。



如果在导入 Unity 插件包的过程中出现下图所示的对话框，点击“I Made a Backup. Go Ahead！”按钮即可。

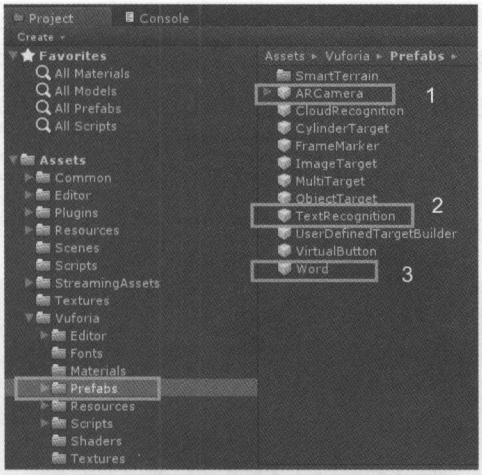


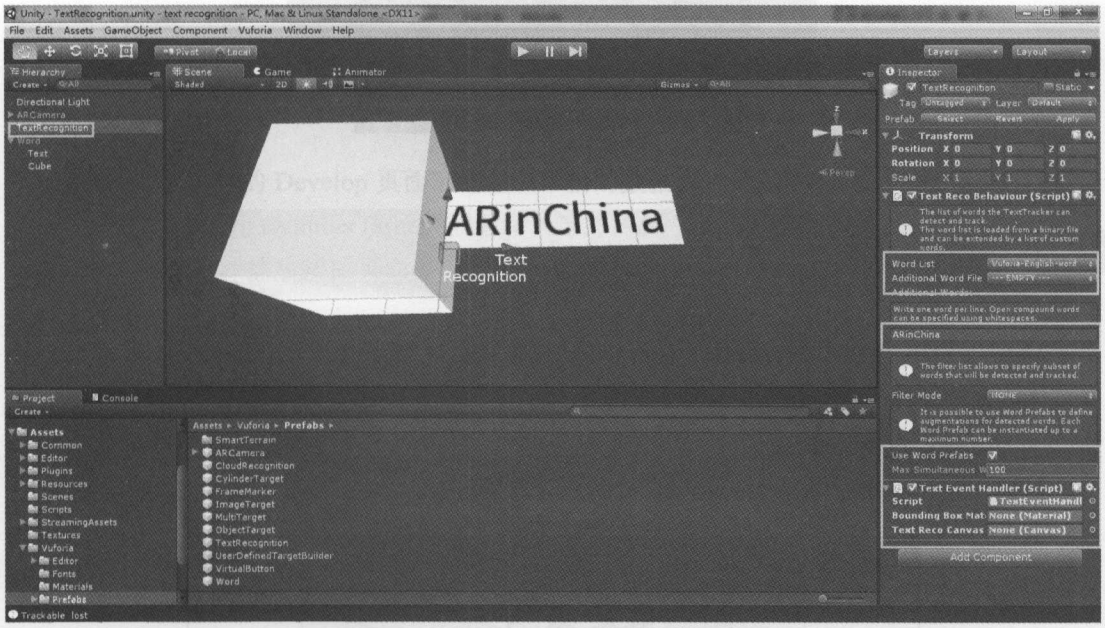
2. 实现文字识别

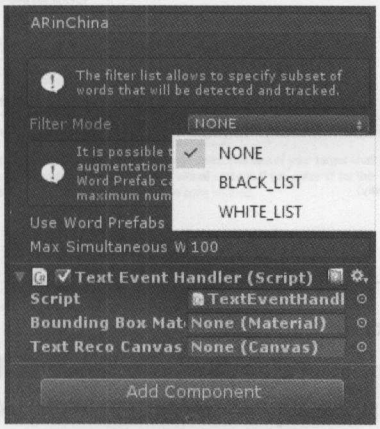
插件包成功导入 Unity 工程后删除场景自带的 Main Camera，同时在 Vuforia 文件夹中找到如下文件，拖到层级视图 Hierarchy 中。

选择场景中新添加的 TextRecognition 物体，在 Inspector 面板中添加 Text Event Handler 脚本。

在 Inspector 面板中，Additional Word File 表示添加自定义词库，类型为 .vwl 文件。上图中，在第二个线框标定区域可以自定义添加需要识别的单词，如 ARinChina。MaxSimultaneous Words 属性是指最大同时存在的物体的单词。


Project
Console
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Common
Editor
Plugins
Resources
Scenes
Scripts
StreamingAssets
Textures
Vuforia
Editor
Fonts
Materials
Prefabs
Resources
Scripts
Shaders
Textures
Assets ▸ Vuforia ▸ Prefabs ▸
SmartTerrain
ARCamera
CloudRecognition
CylinderTarget
FrameMarker
ImageTarget
MultiTarget
ObjectTarget
TextRecognition
UserDefinedTargetBuilder
VirtualButton
Word
1
2
3


Unity - TextRecognition.unity - text recognition - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Vuforia Window Help
Hierarchy
Directional Light
ARCamera
TextRecognition
Word
Text
Cube
Scene
Game
Animator
ARinChina
Text
Recognition
Inspector
TextRecognition
Transform
Position
Rotation
Scale
Text Reco Behaviour (Script)
Word List
Additional Word File
Additional Words
ARinChina
Filter Mode
Use Word Prefabs
Max Simultaneous W 100
Text Event Handler (Script)
Script
Bounding Box Mat
Text Reco Canvas
Add Component
Project
Console
Assets ▸ Vuforia ▸ Prefabs ▸
Trackable lost


ARinChina
The filter list allows to specify subset of words that will be detected and tracked.
Filter Mode
NONE
BLACK_LIST
WHITE_LIST
Use Word Prefabs
Max Simultaneous W 100
Text Event Handler (Script)
Script
TextEventHandl
Bounding Box Mat
None (Material)
Text Reco Canvas
None (Canvas)
Add Component

Black_List 表示黑名单，即将某个单词加入该名单之后就无法再识别；相对应的 White_List 则表示白名单，加入该名单的单词会优先进行识别。

点开 Word，将子对象 Text 文字改为我们自定义的文字 ARinChina，并在场景中的 Word 物体下创建一个识别之后要显示的物体，然后调整其位置和大小，比如创建一个 Cube，最后运行并查看识别效果。

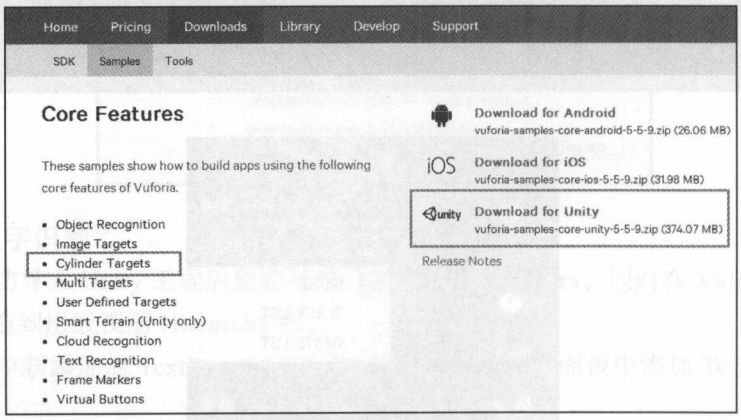


2.5 柱形识别

Vuforia SDK 提供了带有图像的柱形物体识别功能，柱形识别只针对柱形物体，配合包裹在柱体表面的图像来进行识别，比如识别饮料瓶。本节将介绍柱形识别的实现步骤和方法。

1. Vuforia 官方案例包

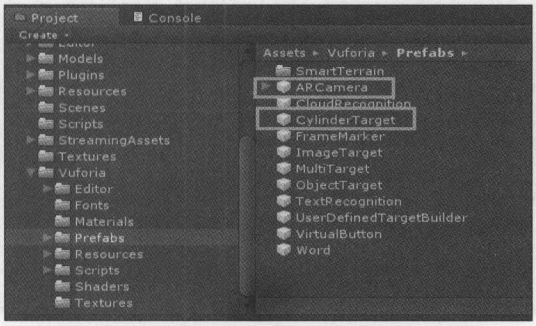
登录 Vuforia 官网，进入 Downloads 页面，点击 Samples 选项，然后下载如下插件并解压。



在解压完成后找到 Cylinder Targets 包并导入 Unity 工程。

在 Project 视图中的 Vuforia 文件夹下找到 ARCamera 预制件和 CylinderTarget 预制件，将这两个预制件拖至层级视图 Hierarchy 中，同时删除场景中的 Main Camera。

CloudReco-5-5-9.unitypackage	2016/3/5 0:58	UNITYPACKAGE ...	39,899 KB
CylinderTargets-5-5-9.unitypackage	2016/3/5 0:59	UNITYPACKAGE ...	37,805 KB
FrameMarkers-5-5-9.unitypackage	2016/3/5 1:00	UNITYPACKAGE ...	35,875 KB
ImageTargets-5-5-9.unitypackage	2016/3/5 1:01	UNITYPACKAGE ...	41,791 KB
MultiTargets-5-5-9.unitypackage	2016/3/5 1:03	UNITYPACKAGE ...	38,318 KB
ObjectRecognition-5-5-9.unitypackage	2016/3/5 1:04	UNITYPACKAGE ...	35,419 KB
SmartTerrain-5-5-9.unitypackage	2016/3/5 1:07	UNITYPACKAGE ...	39,708 KB
TextReco-5-5-9.unitypackage	2016/3/5 1:09	UNITYPACKAGE ...	38,983 KB
UserDefinedTargets-5-5-9.unitypacka...	2016/3/5 1:10	UNITYPACKAGE ...	35,621 KB
VirtualButtons-5-5-9.unitypackage	2016/3/5 1:12	UNITYPACKAGE ...	39,634 KB

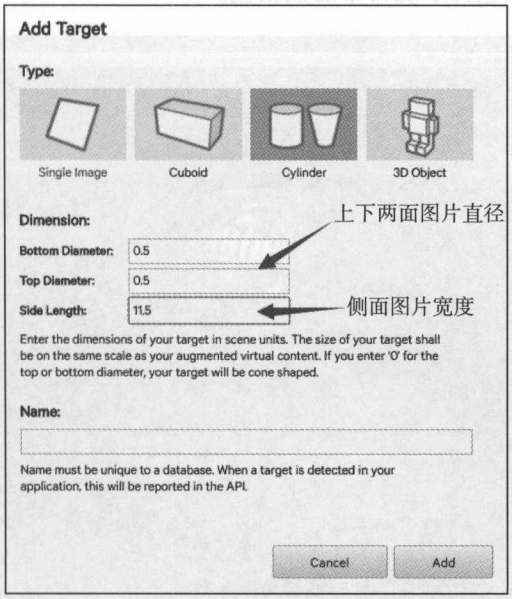


2. 创建数据库

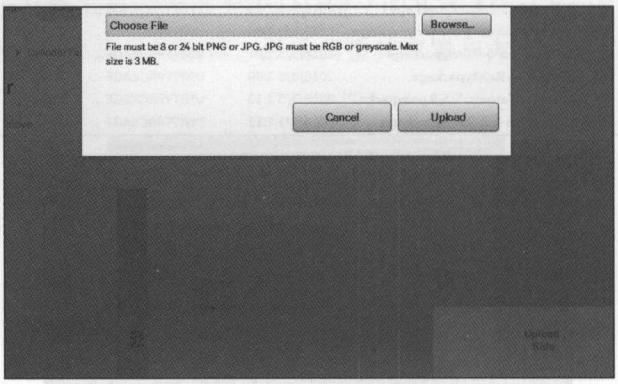
在 Vuforia 官网的 Develop 页面下创建一个新的数据库，命名为 CylinderTargets。

创建完成后打开 CylinderTargets 数据库，选择 Add Target，在弹出的文件选项面板中选择第三种 Cylinder，然后按照页面要求填写圆柱体的尺寸。其中的三个属性分别代表圆柱体的以下信息：

- Bottom Diameter：底面直径。
- Top Diameter：顶面直径。
- Side Length：边长。



在创建好识别目标后，点击新创建的识别目标，然后选择 Upload Image 来上传图片，这个图片是指包裹在圆柱体上的识别图案。



图片上传标准：

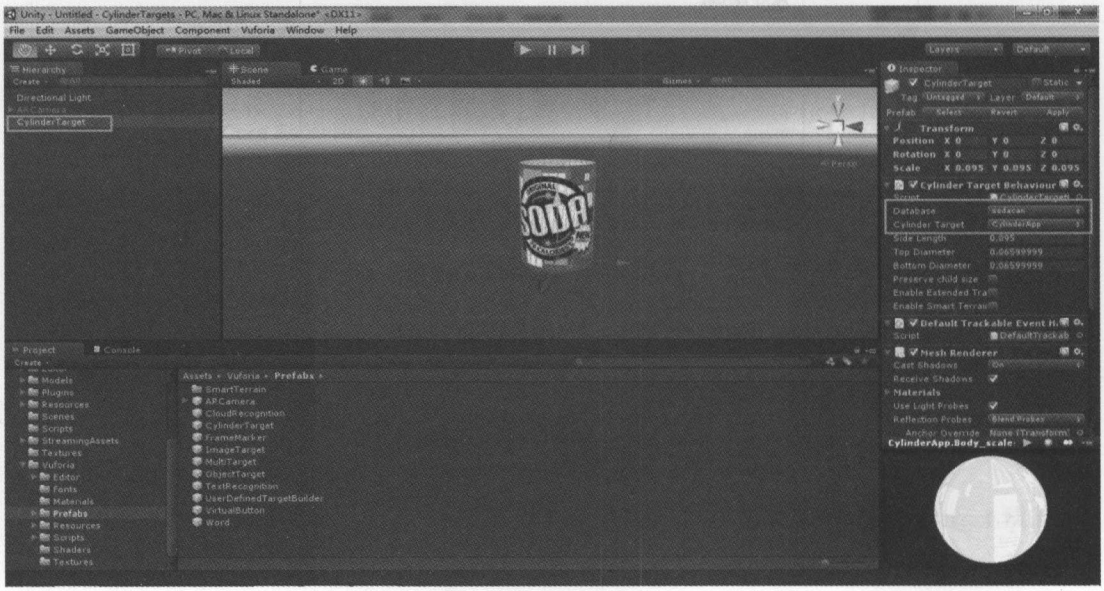
- 长度：Top Diameter（或者 Bottom Diameter）× π。
- 宽度：按照具体圆柱高度确定，本书使用可口可乐易拉罐，所以宽度为易拉罐高度 11.5cm。

注意

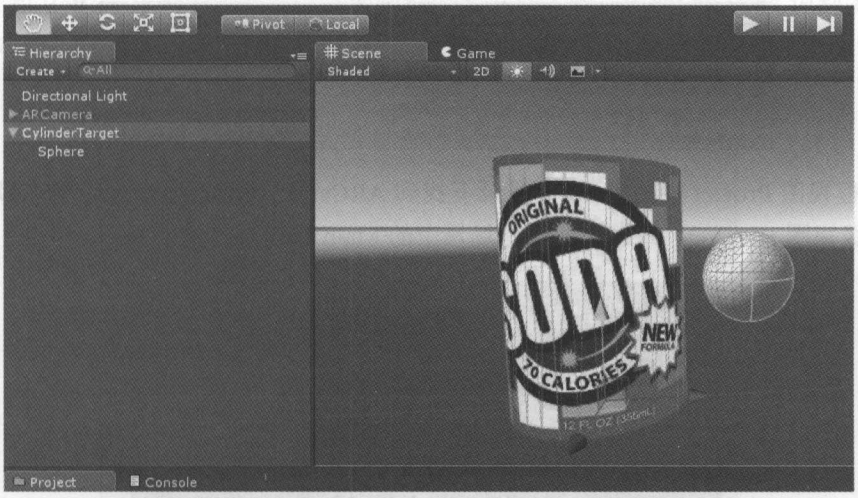
上传图片的长度一定要和上下两面图片的圆形周长相同，图片的宽度即圆柱的高，如果图片尺寸不合适，则无法成功添加。

3. 实现柱形识别功能

这里我们使用官方自带的案例来做示范。在 Unity 工程中选中 CylinderTarget，在 Inspector 面板中选择数据包为系统自带的数据包。



接下来，在 CylinderTarget 下创建一个子物体模型，这里选择 Shpere，然后调节到合适的位置及大小。



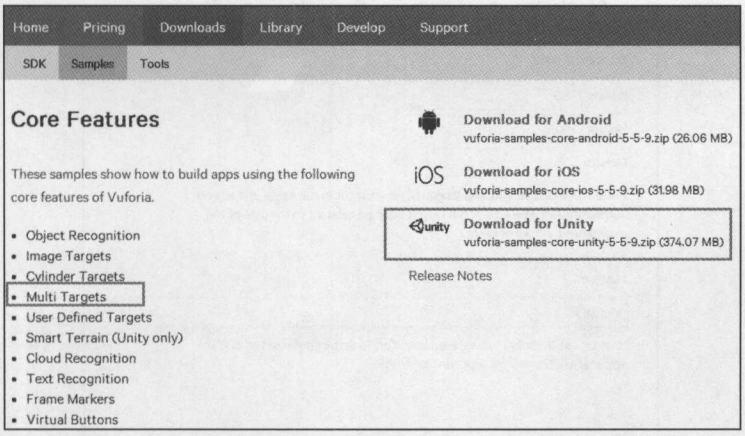
最后，点击运行以测试圆柱体识别的效果。

2.6 立方体识别

Vuforia SDK 提供了立方体识别功能，该功能可以识别被图案包裹的立方体，常用于产品包装盒等物体的 AR 展示效果，本节我们将介绍通过 Unity 以及 Vuforia SDK 制作立方体识别的步骤和方法。

1. 下载并导入插件

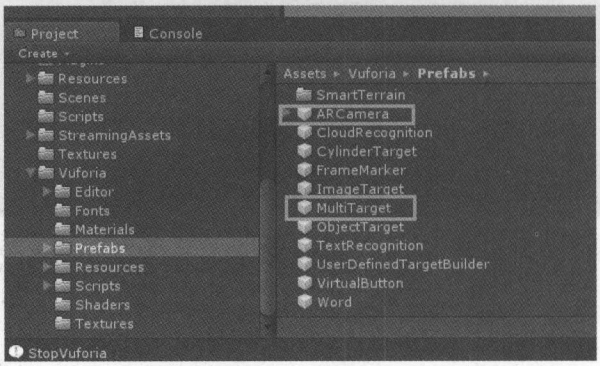
登录官网，点击下载如下插件并解压。



在解压完成后找到 Multi Targets 包并导入 Unity 工程。

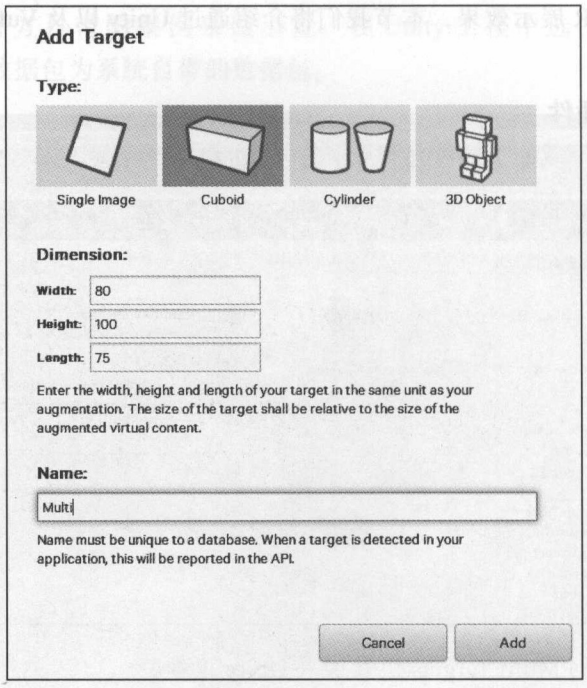
CloudReco-5-5-9.unitypackage	2016/3/5 0:58	UNITYPACKAGE ...	39,899 KB
CylinderTargets-5-5-9.unitypackage	2016/3/5 0:59	UNITYPACKAGE ...	37,805 KB
FrameMarkers-5-5-9.unitypackage	2016/3/5 1:00	UNITYPACKAGE ...	35,875 KB
ImageTargets-5-5-9.unitypackage	2016/3/5 1:01	UNITYPACKAGE ...	41,791 KB
MultiTargets-5-5-9.unitypackage	2016/3/5 1:03	UNITYPACKAGE ...	38,318 KB
ObjectRecognition-5-5-9.unitypackage	2016/3/5 1:04	UNITYPACKAGE ...	35,419 KB
SmartTerrain-5-5-9.unitypackage	2016/3/5 1:07	UNITYPACKAGE ...	39,708 KB
TextReco-5-5-9.unitypackage	2016/3/5 1:09	UNITYPACKAGE ...	38,983 KB
UserDefinedTargets-5-5-9.unitypacka...	2016/3/5 1:10	UNITYPACKAGE ...	35,621 KB
VirtualButtons-5-5-9.unitypackage	2016/3/5 1:12	UNITYPACKAGE ...	39,634 KB

在工程视图 Project 中的 Vuforia 目录下找到 ARCamera 和 MultiTargets 两个预制件，将其拖到层级视图 Hierarchy 中，同时删除场景中的 Main Camera。

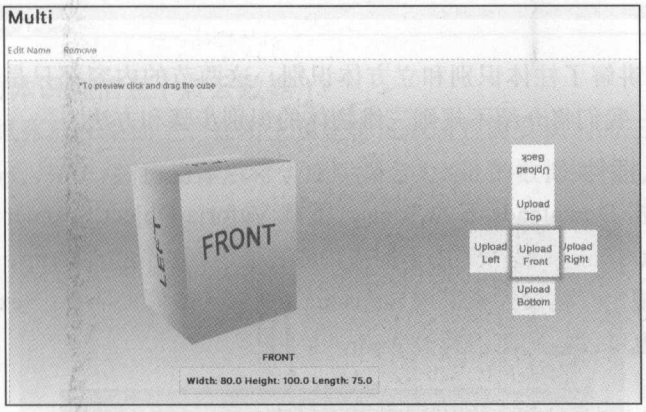


2. 创建数据库

在官网的 Develop 页面下选择 Target Manager 并创建一个新的 Database，命名为 MultiTarget。点击新建的 MultiTarget 数据库，在弹出的文件选项面板中选择 Cuboid。



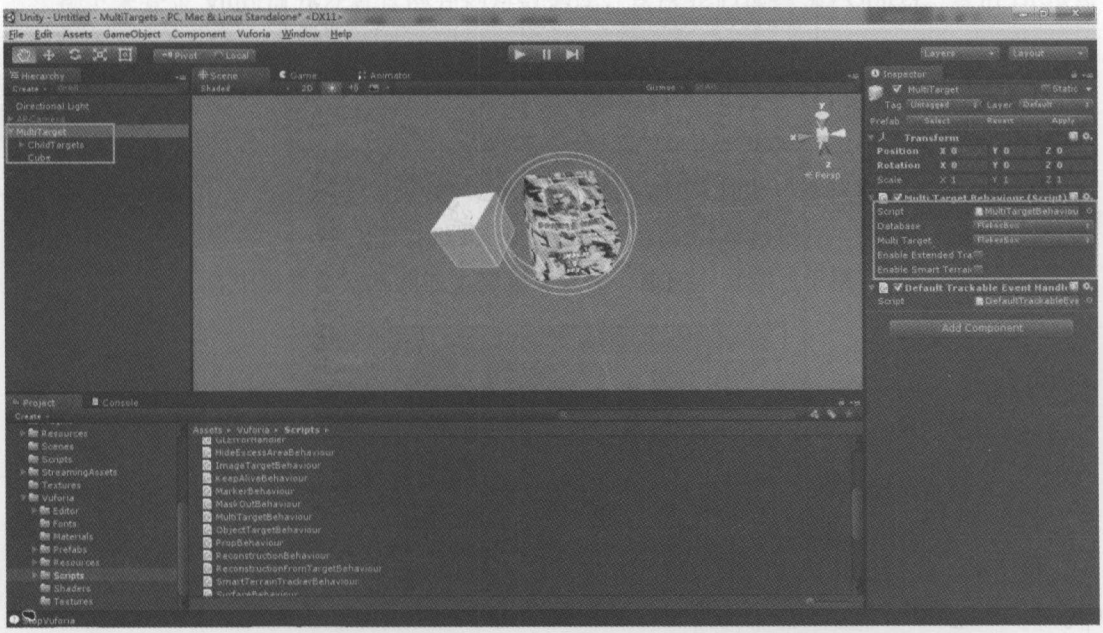
上图中，Dimension 中的属性是为了确定立方体的尺寸，分别为宽、高、长三个维度的尺寸。相应的，物体的长度对应 Front、Bottom、Top 和 Back 识别图的长度，宽度对应 Left、Top、Right 和 Bottom 识别图的宽度，高度则对应 Front、Left、Right 和 Back 识别图的宽度。遵循上述步骤，各识别图上传时相对应位置的尺寸必须一致，否则会出现识别图无法上传的问题。





3. 实现立方体识别功能

这里我们使用官方自带的 MultiTarget，选中 MultiTarget，调节下图所示属性。



在 MultiTarget 下创建一个扫描后要显示的模型，我们可以创建一个 Unity 自带的 Cube 来显示识别之后的物体，调整好对应关系后点击运行即可实现立方体识别效果。

2.7 3D 物体识别

2.5 节和 2.6 节讲解了柱体识别和立方体识别，这两节的内容都只是针对简单几何体进行识别，在本节中，我们将介绍不规则三维物体的识别步骤和方法。

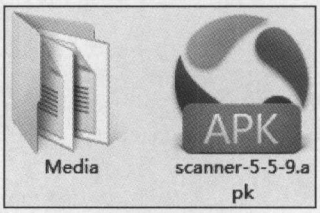
1. 下载并导入插件

登录官网，点击 Downloads 下的 Tools，下载 Vuforia Object Scanner（Vuforia 物体扫描器）。Vuforia Object Scanner 是一个用于扫描物体表面识别点信息的 App，只支持 Android 操作系统。根据官网规定，建议大家采用 Samsung Galaxy S5 和 Google Nexus 5 这两款安卓手机进行操作，当然也可以尝试使用其他 Android 手机。

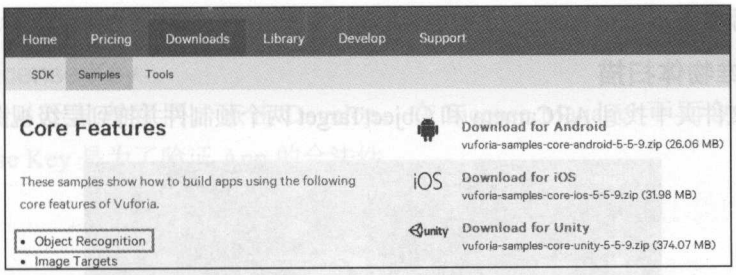


下载完毕并解压后会得到以下两部分内容：

- Media 文件夹：三维物体扫描图，需要采用 A4 纸打印出来。
- scanner.apk：用于三维物体扫描，需要安装在上述两款安卓手机中。

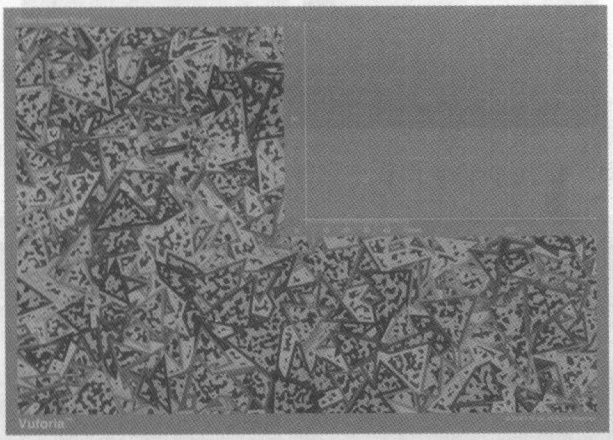


接下来，我们依然按照之前介绍的方法下载 Samples 插件包并将其导入工程，下图中的 Object Recognition 表示对应的插件包中包含物体识别案例。



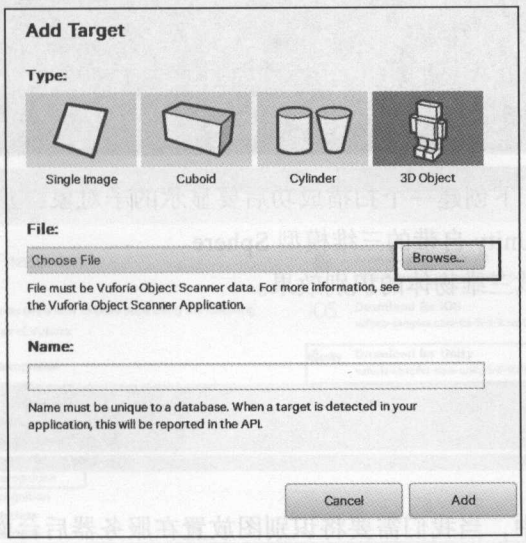
2. 扫描三维物体识别信息（点云数据）

将打印出来的纸张放在桌面上，在右上角坐标区域放置需要识别的三维物体。打开已经安装的扫描数据 App，360° 扫描物体后将生成后缀名为 .od 的数据文件。



3. 创建数据库

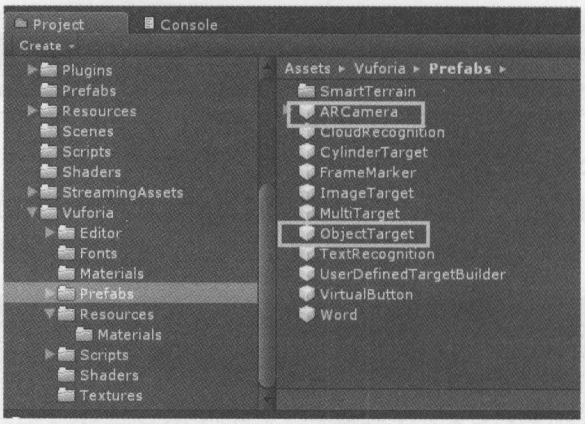
接下来登录 Vuforia 开发者官网并创建数据库，这次我们选择 3D Object，点击 Browse 按钮选择上一步中生成的数据文件。



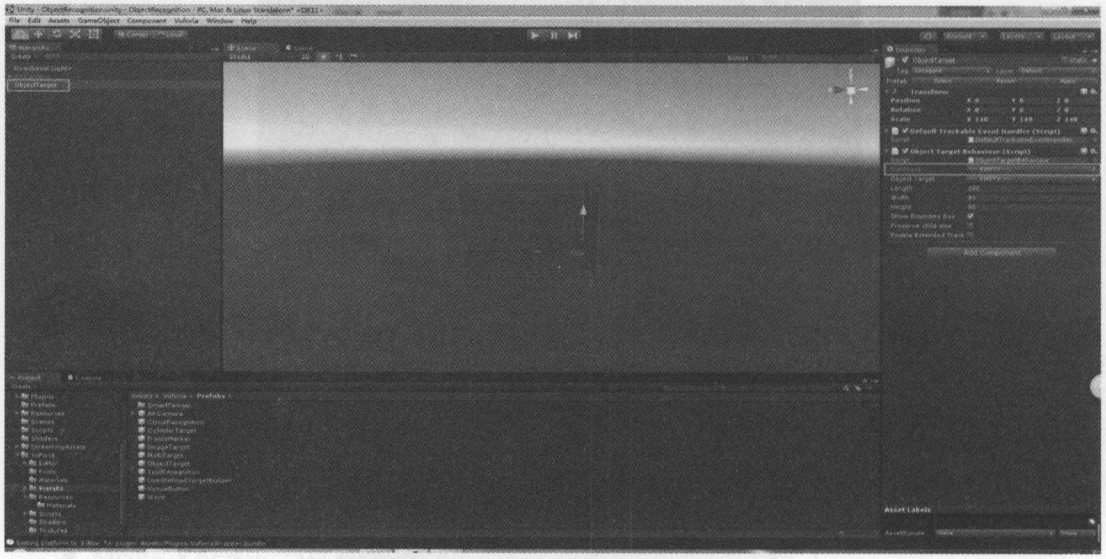
创建成功后点击下载数据包并将其导入 Unity 工程。

4. 实现三维物体扫描

在 Vuforia 文件夹中找到 ARCamera 和 ObjectTarget 两个预制件并拖到层级视图 Hierarchy 中。



选择 ObjectTarget，在 Inspector 面板中选择三维数据包。



然后在 ObjectTarget 下创建一个扫描成功后要显示的子对象，并调节到合适的位置及大小，在此我们直接创建 Unity 自带的三维模型 Sphere。

最后点击运行以测试三维物体的识别效果。

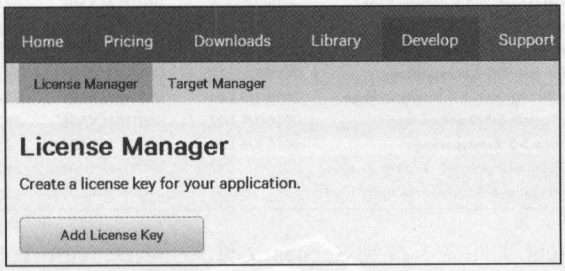
2.8 云识别

在开发项目的过程中，当我们需要将识别图放置在服务器后台而不是 App 内部的时候，

需要使用 Vuforia 的云识别功能，在本节中，我们将介绍 Vuforia 云识别的实现步骤和方法。

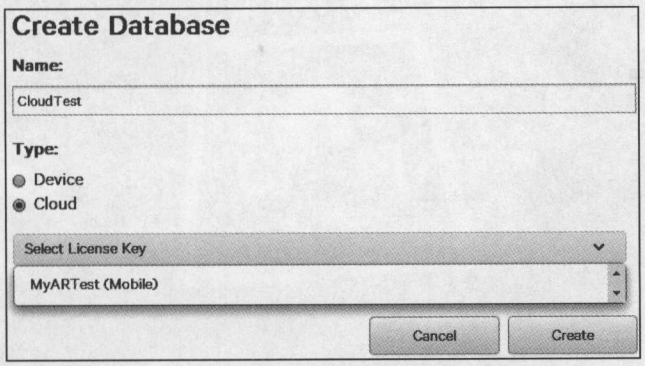
1. 创建 License Key

登录 Vuforia 开发者官网，点击 Develop，在 License Manager 下面创建一个 License Key，该 License Key 是为了验证 App 的合法性。



2. 创建云数据库

选择上图中的 Target Manager 选项，然后点击 Add Database 来创建云识别数据库，在下图中选择 Cloud 即可添加云识别目标。

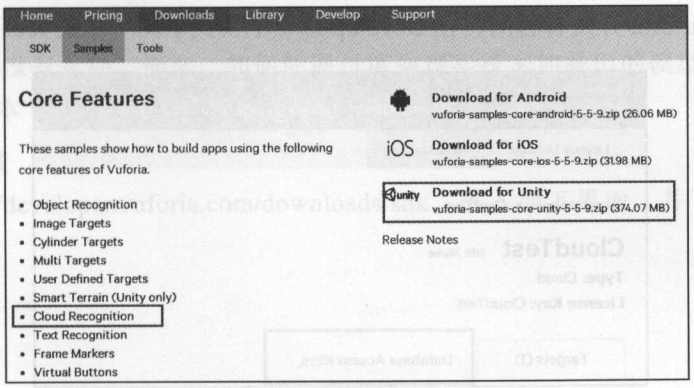


选择刚才创建的 License Key。

创建好之后打开，选择添加图片（与本地图片识别一样）。

3. 下载插件

点击官网中的 Downloads，下载下图所示的 Unity 包。



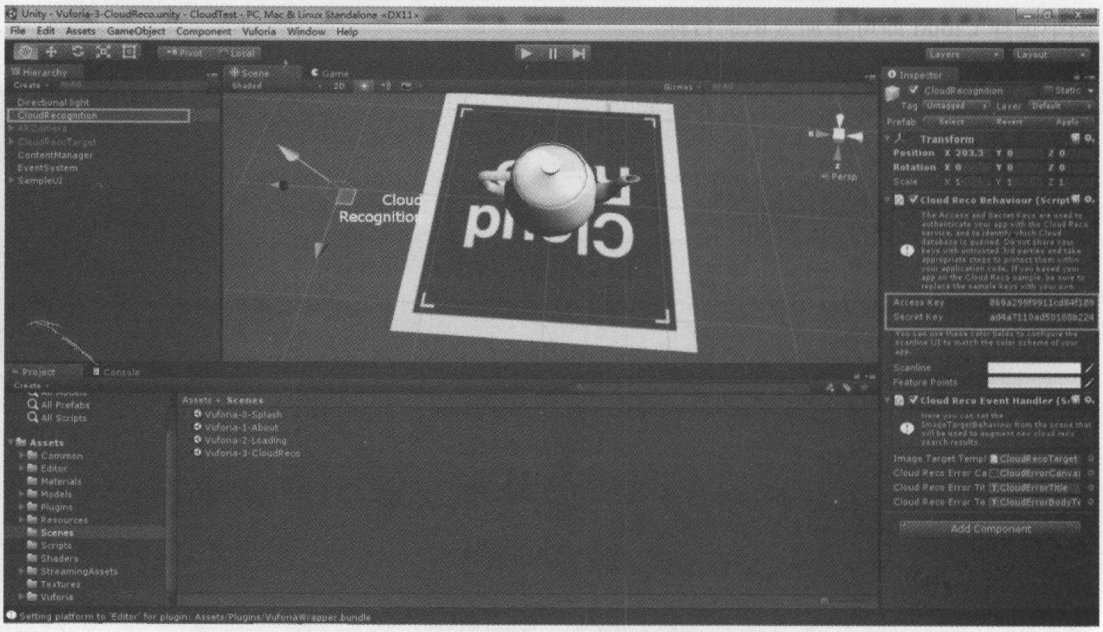
4. 创建 Unity 工程

新建一个 Unity 工程，命名为 CloudTest 并打开，解压下载好的插件包，找到 Cloud Recognition 包并将其导入 Unity 工程。

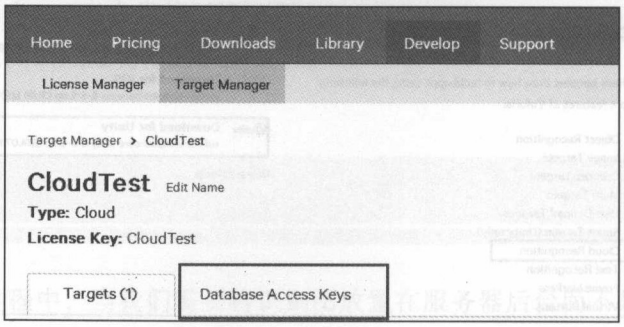
CloudReco-5-5-9.unitypackage	2016/3/5 0:58	UNITYPACKAGE ...	39,899 KB
CylinderTargets-5-5-9.unitypackage	2016/3/5 0:59	UNITYPACKAGE ...	37,805 KB
FrameMarkers-5-5-9.unitypackage	2016/3/5 1:00	UNITYPACKAGE ...	35,875 KB
ImageTargets-5-5-9.unitypackage	2016/3/5 1:01	UNITYPACKAGE ...	41,791 KB
MultiTargets-5-5-9.unitypackage	2016/3/5 1:03	UNITYPACKAGE ...	38,318 KB
ObjectRecognition-5-5-9.unitypackage	2016/3/5 1:04	UNITYPACKAGE ...	35,419 KB
SmartTerrain-5-5-9.unitypackage	2016/3/5 1:07	UNITYPACKAGE ...	39,708 KB
TextReco-5-5-9.unitypackage	2016/3/5 1:09	UNITYPACKAGE ...	38,983 KB
UserDefinedTargets-5-5-9.unitypackage	2016/3/5 1:10	UNITYPACKAGE ...	35,621 KB
VirtualButtons-5-5-9.unitypackage	2016/3/5 1:12	UNITYPACKAGE ...	39,634 KB

导入完成后在 Assets 文件夹中找到 Scenes 文件夹并打开 Vuforia-3-CloudReco 工程。

选择 CloudRecognition，在 Inspector 面板中找到 Access Key 和 Secret Key，通过这两个密钥可以找到云连接的图片。



再次进入 Vuforia 开发者官网，在 Develop 选项下找到下图所示的页面选项。



Client Access Keys

Access Key:

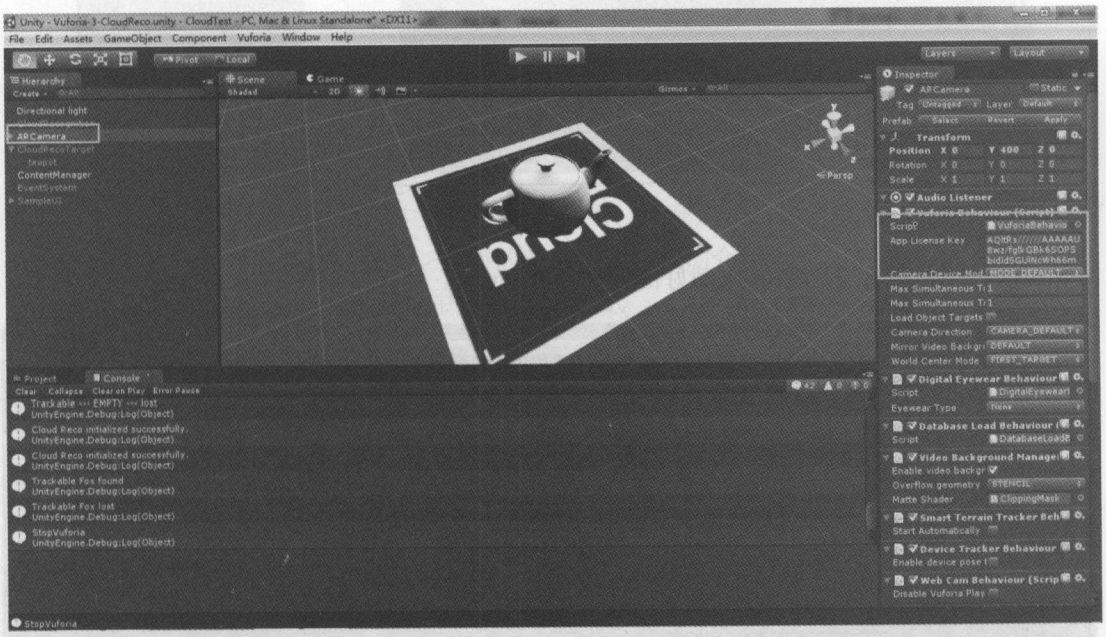
```
dd0d16d81ccd1004c21c2fe8bf4492a70d447650
```

Secret Key:

```
3e34df68d3e875cb92aad9ba7dabbcc93599e2b1
```

将上图中的两个密钥分别复制到 Unity 工程中要求输入密钥的地方。

选择 ARCamera，将刚才专为云识别创建的 License Key 复制进去。



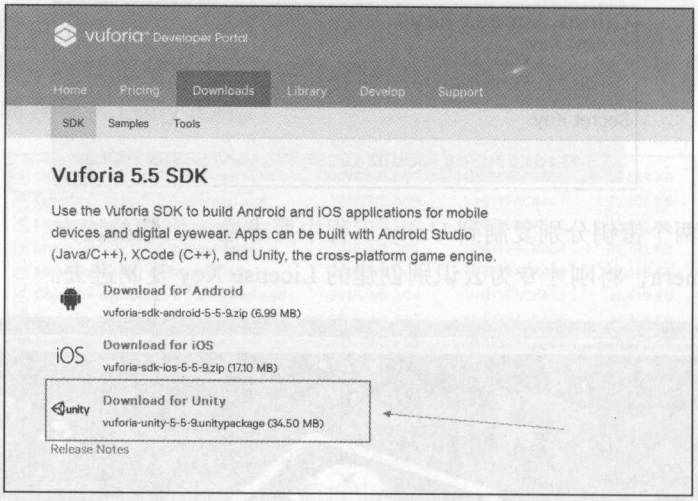
最后运行以测试云识别效果，与本地识别的区别是，云识别测试需要设备处于连网状态。

2.9 智能地形

之前介绍了如何通过扫描单个识别图或物体来实现增强现实，可它们都是相对独立的，并且与现实场景没有互动。那么，如何使通过增强现实技术加载出的模型与现实场景互动呢？这就要用到本节介绍的 SmartTerrain。

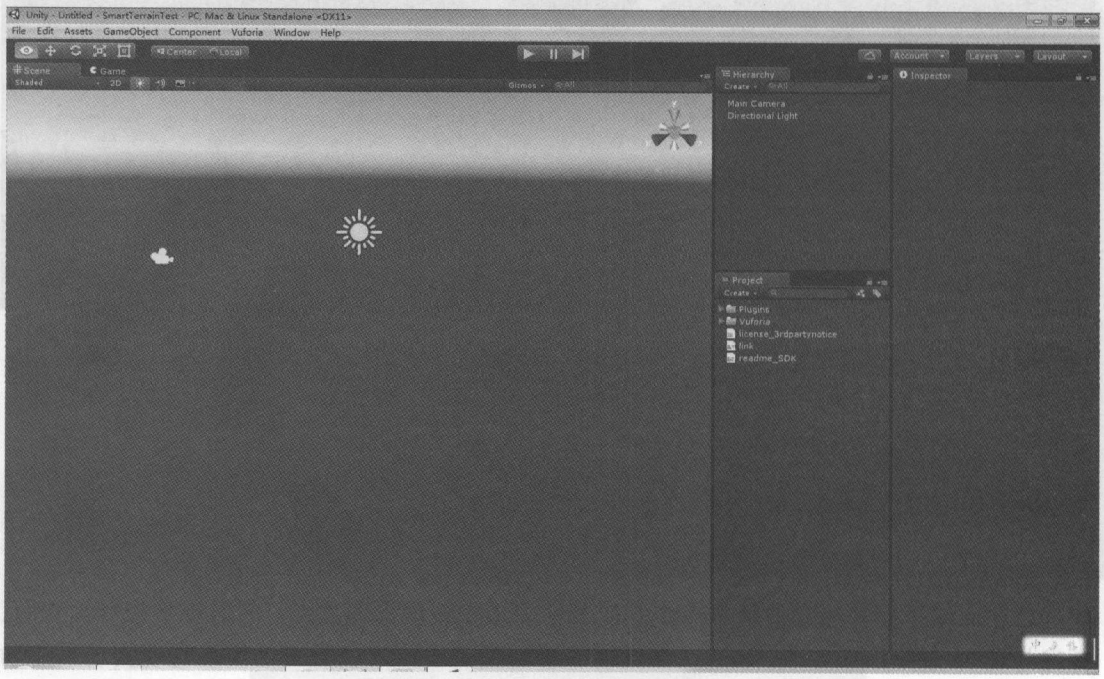
1. 插件下载

访问 https://developer.vuforia.com/downloads/sdk，进入如下界面，并下载 Vuforia SDK 的 Unity 版本。



2. 创建 Unity 工程

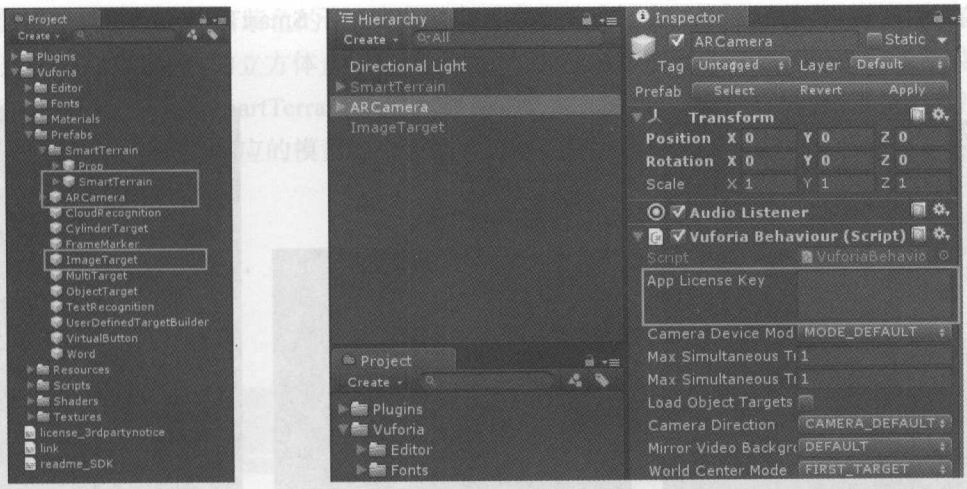
新建一个 Unity 工程并将下载好的插件导入 Unity。



3. 效果实现

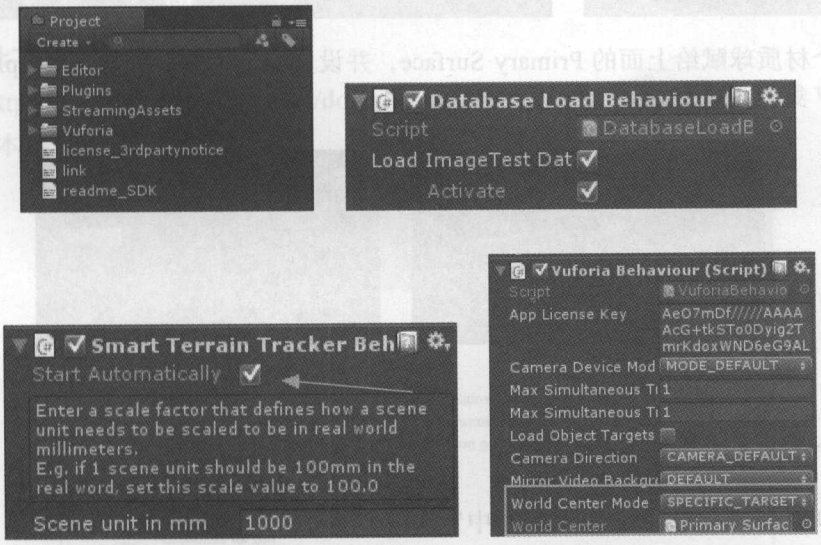
将场景中的 Main Camera 删除，在 Vuforia 文件夹下的 Prefabs 里将 ARCamera、ImageTarget 以及 SmartTerrain 中的 SmartTerrain 拖入场景。

在 Vuforia 的 Develop 页面中的 License Manager 下获取 License Key 并填入下图所示的框内。

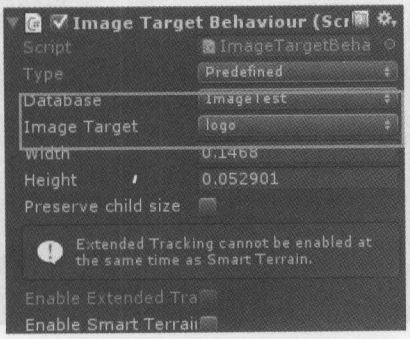


然后在 Target Manager 页面中上传一张识别图，下载识别目标数据文件并将其导入当前工程（关于如何下载识别目标数据，之前的章节中有详细的介绍）。

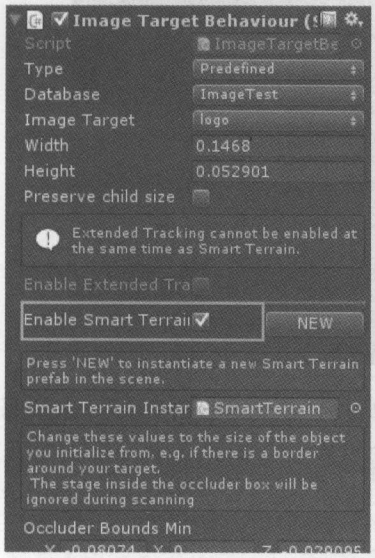
在 ARCamera 的 Inspector 面板中进行如下设置。

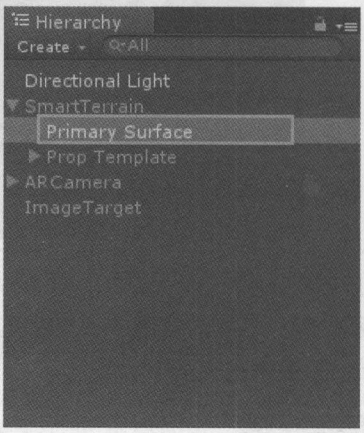


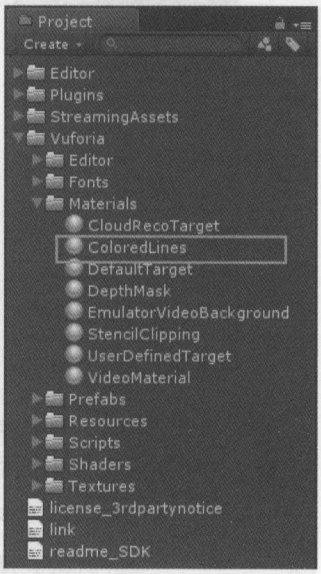
在 ImageTarget 的 Inspector 面板下进行如下设置。



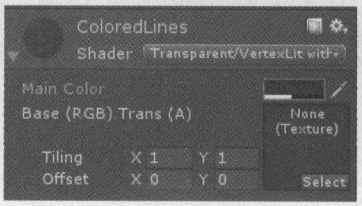
在这里选择你的识别图。为了方便观察，我们改变一下 SmartTerrain 中组件的材质颜色，步骤如下。

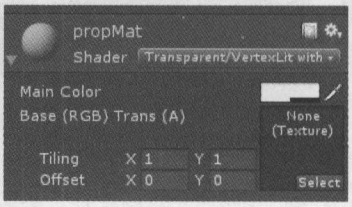






将这个材质球赋给上面的 Primary Surface，并设置如下。然后对 PropTemplate 的材质进行设置。

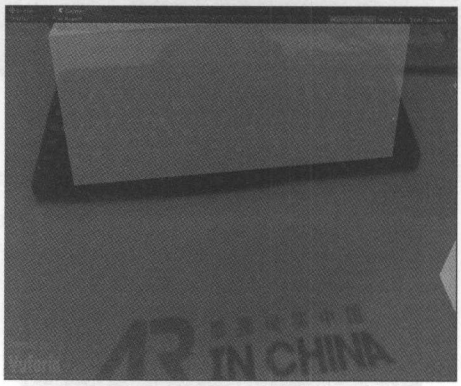




注意

颜色可自由选择，没有特殊规定。

运行 Unity 扫描识别图，并在场景中置入一个随机物体，效果图如下。



扫描成功后会出现暗紫色的阴影部分，随后如果扫描出了场景中的三维物体，就会在该物体上生成一个浅绿色的立方体盒子（不同设置下的颜色不同）。

至此，Vuforia 的 SmartTerrain 地形就创建完毕，大家可以根据自己的需求在出现浅绿色立方体盒子的地方添加相应的模型与交互事件，从而完善你的 SmartTerrain 应用。

注意

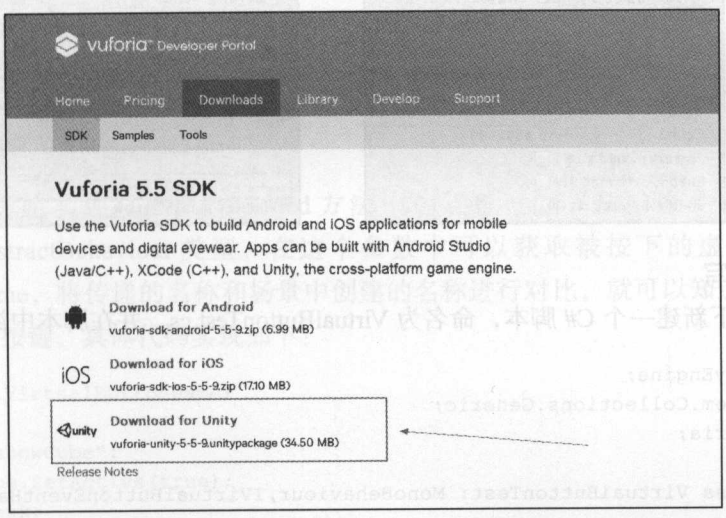
若想实现更好的效果，还可以结合 PrimeSense Capri 进行开发。

2.10 虚拟按钮

使用 Vuforia 实现增强现实之后，有时需要与这些虚拟模型进行交互，同时为了使交互方式更加魔幻，我们期望可以在真实的识别图像上进行点击，从而触发 App 中的某些行为。Vuforia SDK 为我们提供了 Virtual Button 功能来实现这样的交互，本节我们将带领大家使用这一特性实现两个模型之间的切换。

1. 插件下载

访问 https://developer.vuforia.com/downloads/sdk，进入如下界面，并下载 Vuforia SDK 的 Unity 版本。

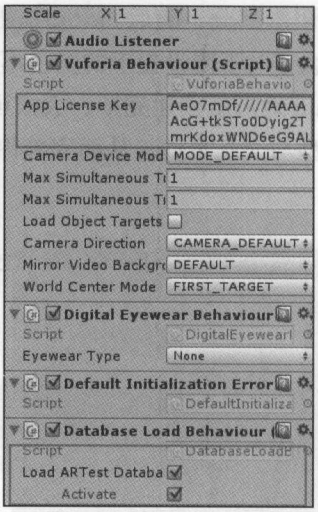


2. 创建 Unity 工程

新建一个 Unity 工程并将下载好的插件导入 Unity，同时将识别图数据包导入该工程。

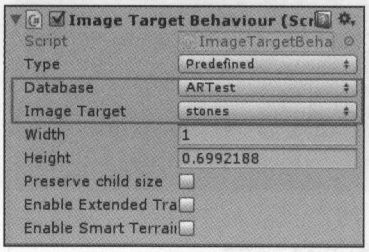
3. 场景搭建

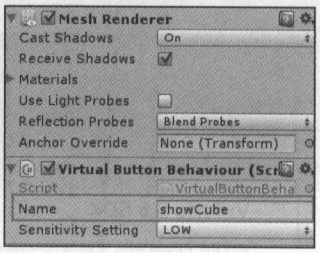
首先删除场景中的 Main Camera，接着从 Vuforia → Prefabs 中将 ARCamera 和 ImageTarget 拖入场景，在 ARCamera 的 Inspector 面板下添加 App License Key，并激活当前识别图信息。



然后在 ImageTarget 的 Inspector 面板下设置相应识别图信息。

接着在 ImageTarget 下分别创建一个 Cube 和一个 Sphere，并将 Project 视图中 Vuforia → Prefabs 下的 Virtual Button 预制件拖入 ImageTarget。这里将 Cube 的棱长设置为 0.3，将 Sphere 的半径设置为 0.4，然后使它们的位置相同。为了便于观察，我们新建两个材质球，分别将 Cube 和 Sphere 设置为不同的颜色。最后在 Virtual Button 的 Inspector 界面里分别设置两个 Virtual Button 的名字。





4. 脚本编写

在 Project 下新建一个 C# 脚本，命名为 VirtualButtonTest.cs，并在脚本中添加以下代码：

```
using UnityEngine;
using System.Collections.Generic;
using Vuforia;

public class VirtualButtonTest: MonoBehaviour,IVirtualButtonEventHandler
{
    void Start ()
    {
    }

    public void OnButtonPressed(VirtualButtonAbstractBehaviour vb) { }

    public void OnButtonReleased(VirtualButtonAbstractBehaviour vb) { }
}
```

该类的作用是实现 IVirtualButtonEventHandler 这个接口，该接口中定义了 OnButtonPressed 和 OnButtonReleased 两个方法，这两个方法能够监听虚拟按键的按下和释放事件。为了触发这两个事件，我们还需要将之前添加的两个 VirtualButton 对象注册到事件系统中，因此需要在 Start（）方法中查找这两个物体，并调用 VirtualButtonBehaviour 类中的 RegisterEventHandler 方法将当前对象作为参数传入，具体的代码实现如下：

```
void Start ()
{
    // 在所有子物体类中找到所有 VirtualButtonBehaviour 组件
    VirtualButtonBehaviour[] vbs = GetComponentsInChildren<VirtualButtonBehaviour>();
    for (int i = 0; i < vbs.Length; ++i)
    {
        // 在虚拟按钮中注册 TrackableBehaviour 事件
        vbs[i].RegisterEventHandler(this);
    }
}
```

接下来在当前类中创建两个 GameObject 类型的字段，分别对应之前创建的立方体和球体。

```
private GameObject cube;
private GameObject sphere;
```

接着在 Start 方法中查找这两个物体并赋值给相应的字段。

```
cube = transform.FindChild("Cube").gameObject;
sphere = transform.FindChild("Sphere").gameObject;
```

在程序开始的时候，我们希望它们都不显示，所以要先把它们藏起来。这就是 Start 方法的最后一步需要做的事情：

```
cube.SetActive(false);
sphere.SetActive(false);
```

现在我们开始实现 OnButtonPressed 方法中的功能，此方法的唯一一个参数 vb 是 VirtualButtonAbstractBehaviour 类型。在这个参数中可以获取被按下的虚拟按键的名称 VirtualButtonName，将传递的名称和场景中创建的名称进行对比，就可以知道我们具体按下的是哪一个虚拟按键，具体代码实现如下：

```
switch (vb.VirtualButtonName)
{
    case "showCube":
        cube.SetActive(true);
        break;
    case "showSphere":
        sphere.SetActive(true);
        break;
}
```

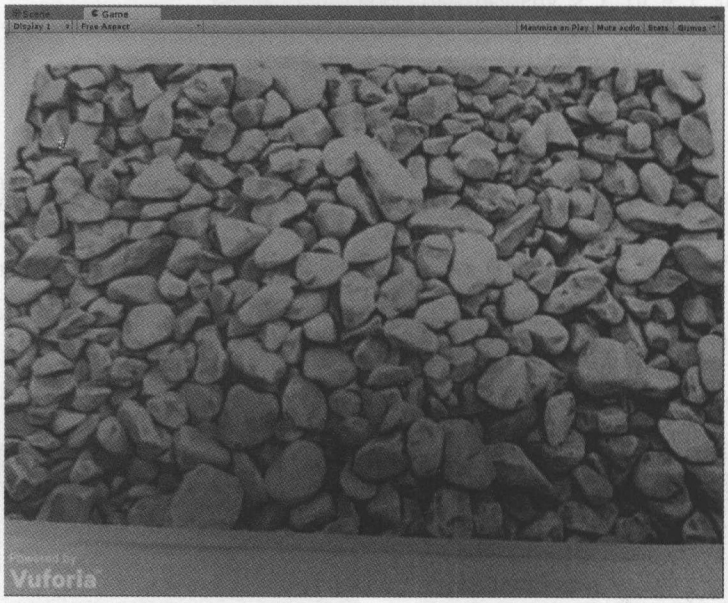
同理，OnButtonReleased 实现方法基本相同，代码如下：

```
switch (vb.VirtualButtonName)
```

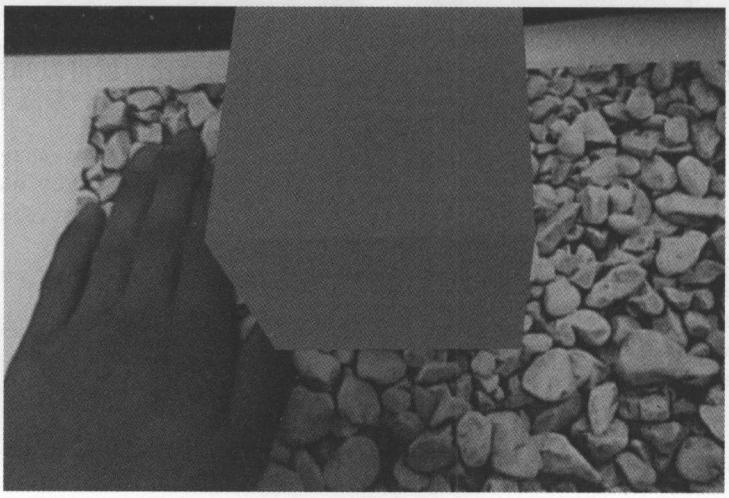
```
{
    case "showCube":
        cube.SetActive(false);
        break;
    case "showSphere":
        sphere.SetActive(false);
        break;
}
```

5. 功能实现

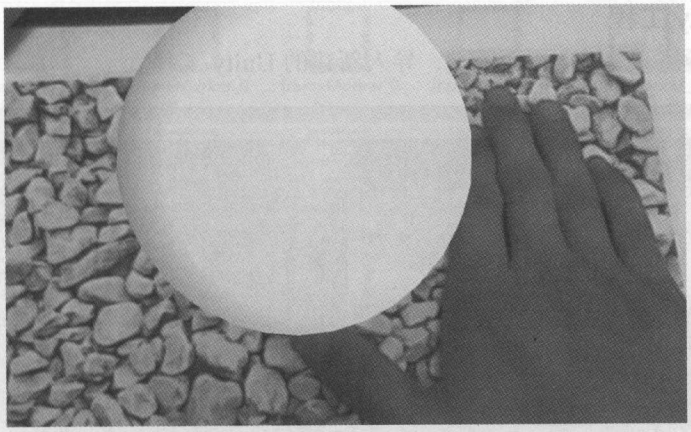
运行 Unity 并扫描识别图，由于我们在最开始将两个物体隐藏起来了，所以看不到东西。



当我们“按下”左边的 showCube 按钮时，画面如下。



接着，我们“按下”右边的 showSphere 按钮，效果如下。



到此，VirtualButton功能实现完毕。

关于虚拟按钮，有以下几点需要注意：

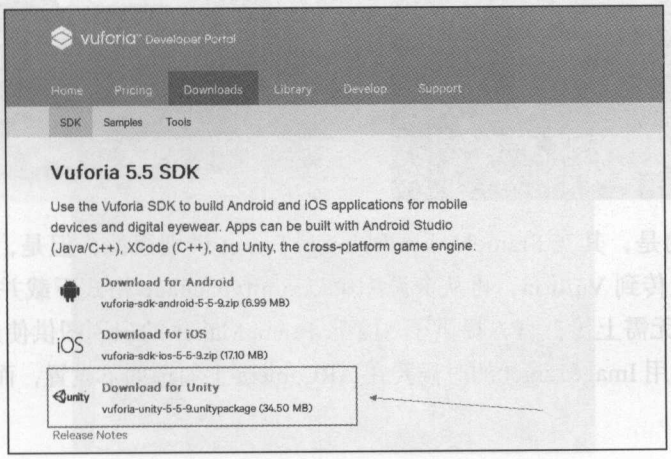
- 本节只是实现了一个简单的功能，大家可以根据自己的需求自定义VirtualButton事件。
- 可以使用PS等图像处理软件给识别图加上虚拟按钮的图像，这样会更直观。
- 为了达到更好的体验效果，识别图可以复杂一些，尤其是在虚拟按钮的位置。

2.11 帧标记识别

在使用Vuforia进行开发时，我们总是强调识别图的识别度星级越高，越容易识别。那么假如在商业项目中，客户要使用的识别图的识别星级为0，该怎么办呢？这就是FrameMarker大展拳脚的时候了。本节我们将为大家介绍一个名叫FrameMaker的神奇功能。

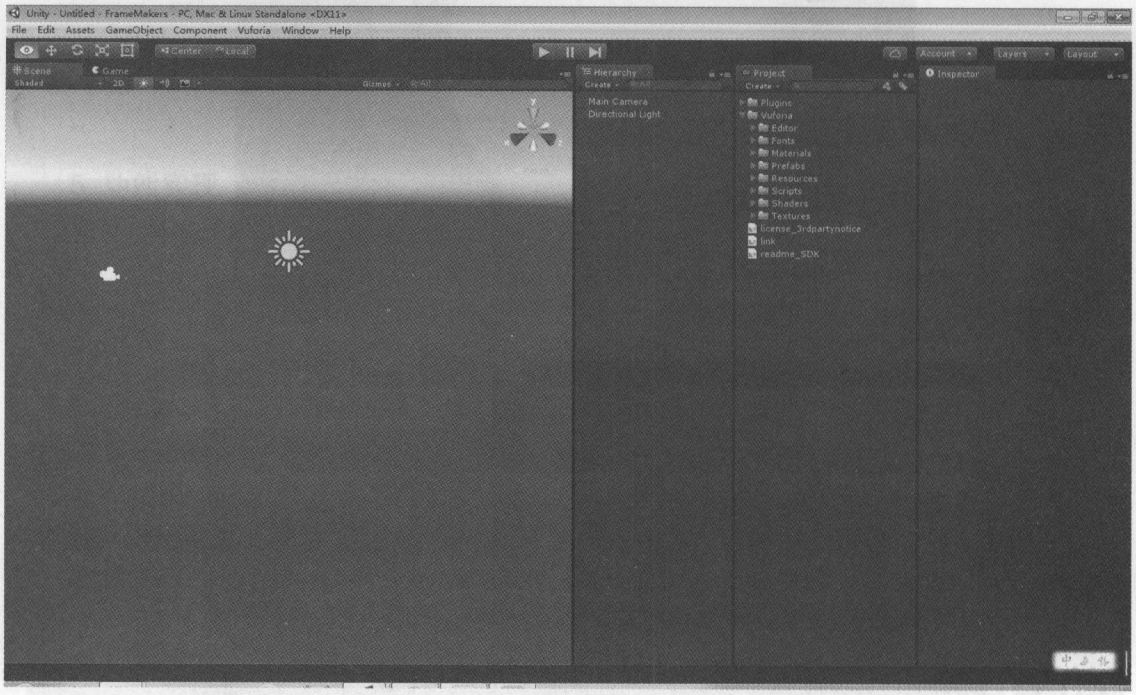
1. 插件下载

访问https://developer.vuforia.com/downloads/sdk，进入如下界面，并下载Vuforia SDK的Unity版本。



2. 创建 Unity 工程

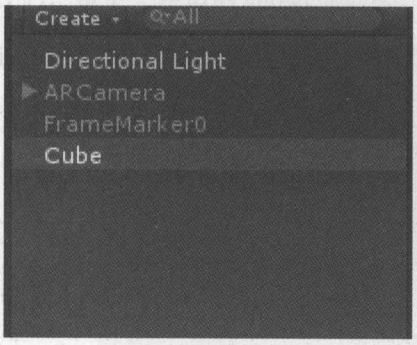
使用插件 vuforia-unity-5-5-9.unitypackage，导入新建的 Unity 工程。

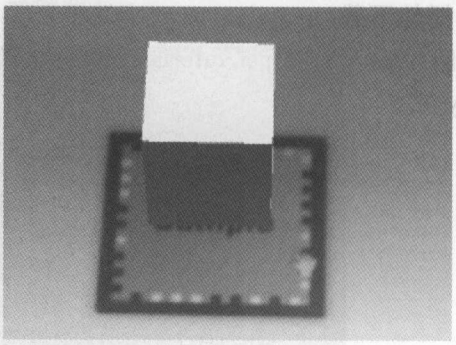


然后拖入 ARCamera 和 FrameMaker，并在 FrameMaker 下新建一个 Cube 作为识别后呈现的对象。

3. 效果实现

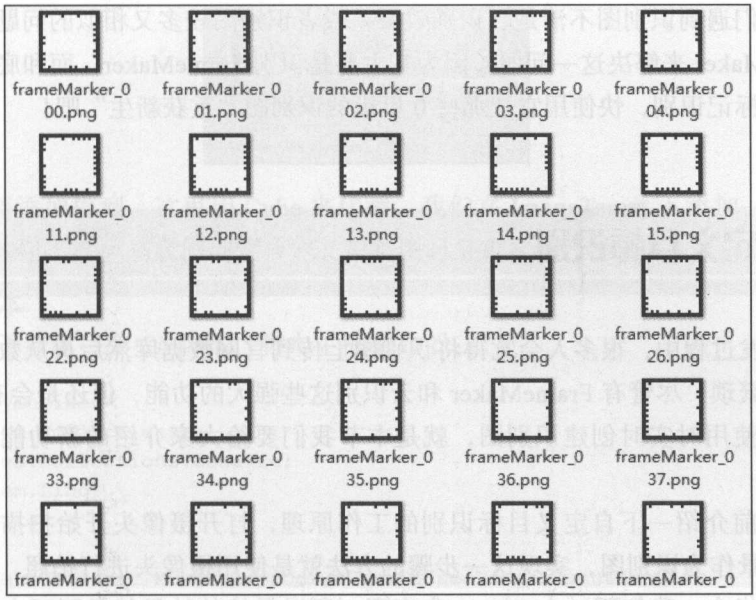
在 ARCamera 里填写 App License Key，识别效果如下。

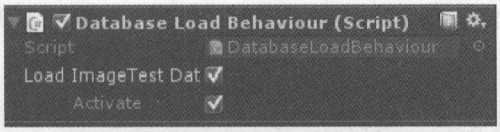




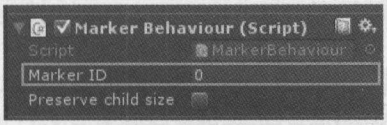
这里要说明的是，其实 FrameMaker 和 ImageTarget 识别很像，但是，ImageTarget 识别需要先将识别图上传到 Vuforia，再从资源库中以 .unitypackage 格式下载并导入工程中使用。而 FrameMaker 则无需上传，官方提供了 512 张 FrameMaker 的识别图供使用。

所以当我们使用 ImageTarget 的时候要在 ARCamera 上勾选如下设置，而使用 FrameMaker 的时候则无需设置。

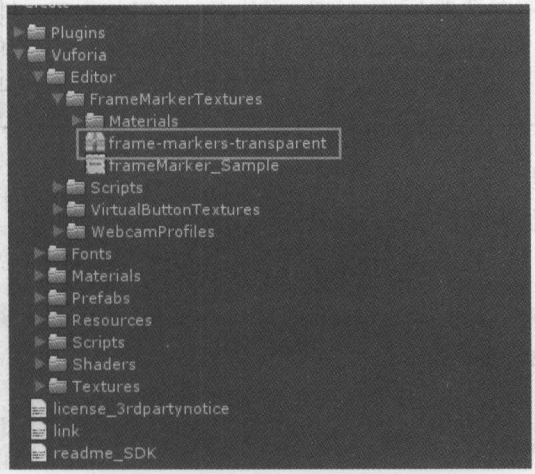




所以我们在制作的时候，只需使用 PS 将自己的识别图和官方提供的 FrameMaker 进行合成即可，因为所识别的是帧识别图，与底图内容无关。但要注意一点，你所使用的帧识别图的序号需在场景中的 FrameMaker 中进行设定，默认为 0（即第一张）。



官方提供的识别图已打包好，解压后可直接使用，路径如下。



所以当我们遇到识别图不清楚、识别效果差或者识别图较多又相似的问题时，就可以考虑使用 FrameMaker 来解决这一问题，因为其主要是识别 FrameMaker，而和底图没有关系。

这就是帧标记识别，快使用它让那些 0 星级的识别图“重获新生”吧！

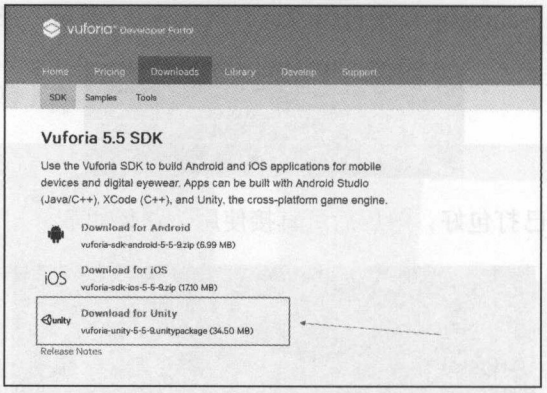
2.12 自定义目标识别

在 AR 开发过程中，很多人会觉得将识别图上传到官网数据库然后再从数据库下载这个过程难免有些繁琐，尽管有 FrameMaker 和云识别这些强大的功能，但还是会有人觉得麻烦。那么，如何在使用时实时创建识别图，就是本节我们要给大家介绍的新功能，自定义目标识别。

有必要提前介绍一下自定义目标识别的工作原理，打开摄像头开始扫描后，以扫描到的某一固定场景作为识别图，实现这一步骤的方法就是使用摄像头进行拍照，这时就需要一个触发拍照的指令，我们用一个 Button 来实现。所以最终的效果就是画面中会一直有一个 Button，当我们扫描到我们要自定义的识别图时，按下 Button，然后识别图创建完成，扫描该识别图，模型出现。

1. 插件下载

访问 https://developer.vuforia.com/downloads/sdk，进入如下界面，并下载 Vuforia SDK 的 Unity 版本。



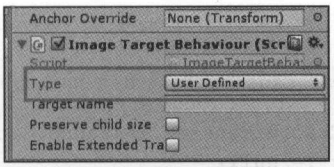
2. 创建 Unity 工程

新建一个 Unity 工程并将下载好的插件导入 Unity。

3. 场景搭建

删除场景中原有的 Main Camera，接着从 Vuforia → Prefabs 路径下将 ARCamera、ImageTarget 以及 UserDefinedTargetBuilder 拖入场景。

接着在 ARCamera 的 Inspector 面板下添加 App License Key。随后在 ImageTarge 的 Inspector 面板下将 ImageTarget 类型设置为 UserDefined。



接下来需要添加模型，这里用 Cube 来代替，我们在 ImageTarget 下添加一个 Cube。最后创建 Button 按钮，这也是我们创建自定义识别图时最重要的一步。

4. 编写脚本

在 Project 下新建一个 C# 脚本，脚本主框架如下：

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Vuforia;

public class UDTTest: MonoBehaviour,IUserDefinedTargetEventHandler
{
    void Start ()
    {

    }
     public void OnFrameQualityChanged(ImageTargetBuilder.FrameQuality
frameQuality) { }
    public void OnInitialized() { }
    public void OnNewTrackableSource(TrackableSource trackableSource) { }
    public void BuildNewTarget()
    {
    }
}
```

这个类中比较关键的是 IUserDefinedTargetEventHandler 这个接口，其中定义了 OnFrame QualityChanged、OnInitialized 和 OnNewTrackableSource 三个方法。我们实现这个接口中的三个方法，并将脚本附加到 UserDefinedTargetBuilder 上。

首先需要声明一个 UserDefinedTargetBuildingBehaviour 类型的变量，接着在 Start 函数里将当前类注册为该类型事件。

```
UserDefinedTargetBuildingBehaviour mTargetBuildingBehaviour;
    void Start ()
    {
        mTargetBuildingBehaviour = GetComponent<UserDefinedTargetBuildingBehaviour>();
        if (mTargetBuildingBehaviour)
        {
            mTargetBuildingBehaviour.RegisterEventHandler(this);
            Debug.Log("Registering User Defined Target event handler.");
        }
    }
```

接下来要实现接口中的三个函数，首先是 OnInitialized，用来初始化一些数据，我们在

这个函数里写入下面的代码：

```
public void OnInitialized()
    {
        mObjectTracker = TrackerManager.Instance.GetTracker<ObjectTracker>();
        if (mObjectTracker != null)
        {
            mBuiltDataSet = mObjectTracker.CreateDataSet();
            mObjectTracker.ActivateDataSet(mBuiltDataSet);
        }
    }
```

这里用到两个新的变量，需要提前声明，如下：

```
ObjectTracker mObjectTracker;
// 新定义的数据集添加到 DataSet 里
DataSet mBuiltDataSet;
```

接着是 OnFrameQualityChanged 这个函数，用来更新当前帧质量。我们在函数里写入下面的代码：

```
public void OnFrameQualityChanged(ImageTargetBuilder.FrameQuality frameQuality)
{
    mFrameQuality = frameQuality;
    if (mFrameQuality == ImageTargetBuilder.FrameQuality.FRAME_QUALITY_LOW)
    {
        Debug.Log("Low camera image quality");
    }
}
```

这里的 mFrameQuality 是一个用来记录当前帧图像质量的枚举类型，声明如下：

```
ImageTargetBuilder.FrameQuality mFrameQuality = ImageTargetBuilder.FrameQuality.
FRAME_QUALITY_NONE;
```

最后也是最重要的一个方法 OnNewTrackableSource 的代码如下：

```
public void OnNewTrackableSource(TrackableSource trackableSource)
{
    mTargetCounter++;
    // Deactivates the dataset first
    mObjectTracker.DeactivateDataSet(mBuiltDataSet);

    // Destroy the oldest target if the dataset is full or the dataset
    // already contains five user-defined targets.
    if (mBuiltDataSet.HasReachedTrackableLimit() || mBuiltDataSet.GetTrackables().
Count() >= 5)
    {
        IEnumerable<Trackable> trackables = mBuiltDataSet.GetTrackables();
        Trackable oldest = null;
        foreach (Trackable trackable in trackables)
        {
            if (oldest == null || trackable.ID < oldest.ID)
                oldest = trackable;
            }
```

```
            if (oldest != null)
            {
                Debug.Log("Destroying oldest trackable in UDT dataset: " + oldest.Name);
                mBuiltDataSet.Destroy(oldest, true);
            }
        }

        // Get predefined trackable and instantiate it
        ImageTargetBehaviour imageTargetCopy = (ImageTargetBehaviour)Instantiate(Imag
eTargetTemplate);
        imageTargetCopy.gameObject.name = "UserDefinedTarget-" + mTargetCounter;

        // Add the duplicated trackable to the data set and activate it
        mBuiltDataSet.CreateTrackable(trackableSource, imageTargetCopy.gameObject);

        // Activate the dataset again
        mObjectTracker.ActivateDataSet(mBuiltDataSet);
    }
```

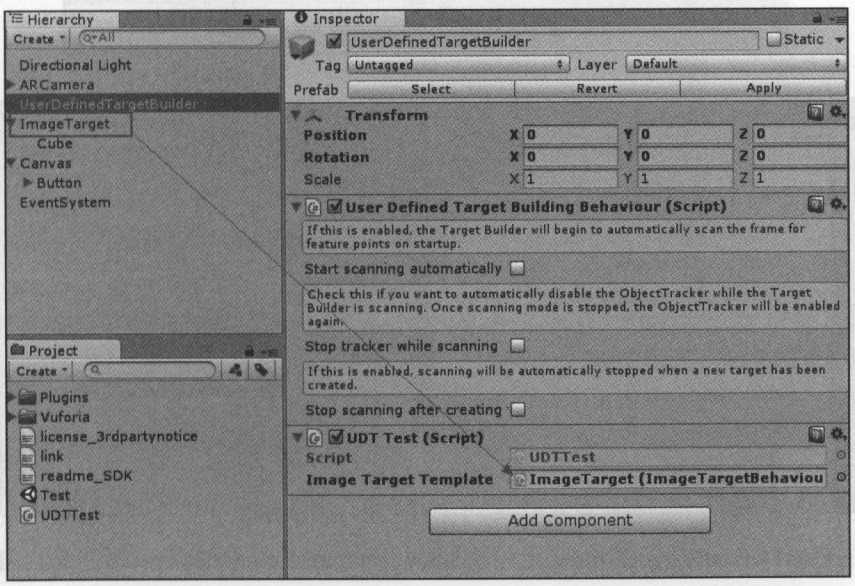
这个函数里需要提前声明的变量有两个，如下：

```
int mTargetCounter;
// 声明一个公开的 ImageTargetBehaviour，然后在 Unity 中赋值
public ImageTargetBehaviour ImageTargetTemplate;
```

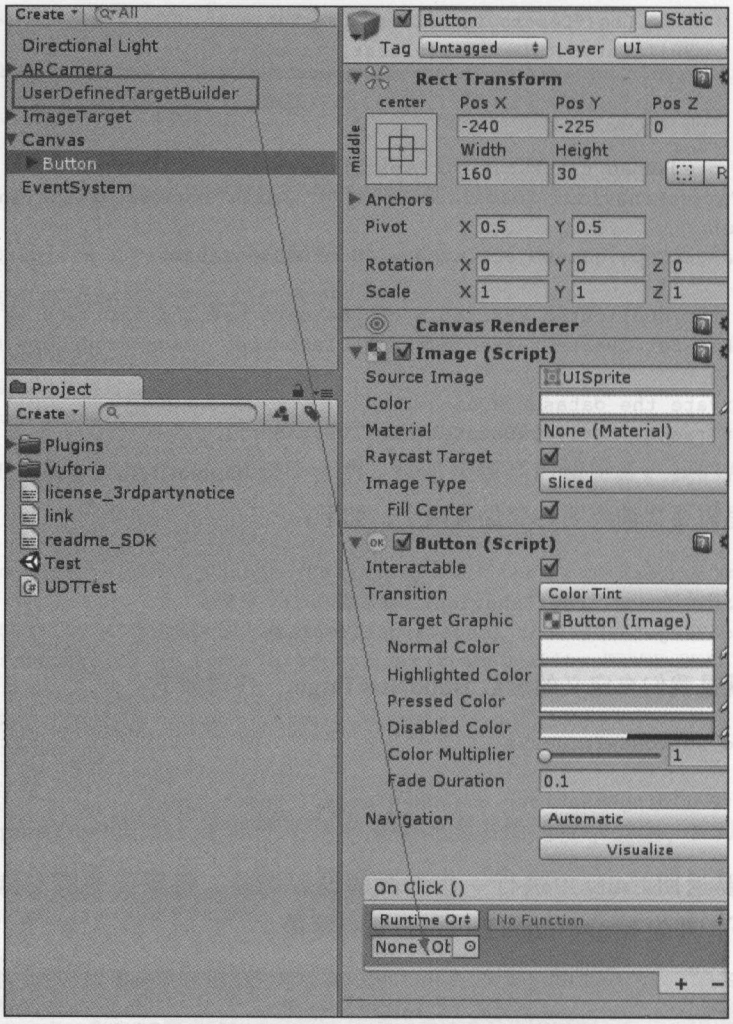
最后一个就是我们自定义的函数 BuildNewTarget，代码如下：

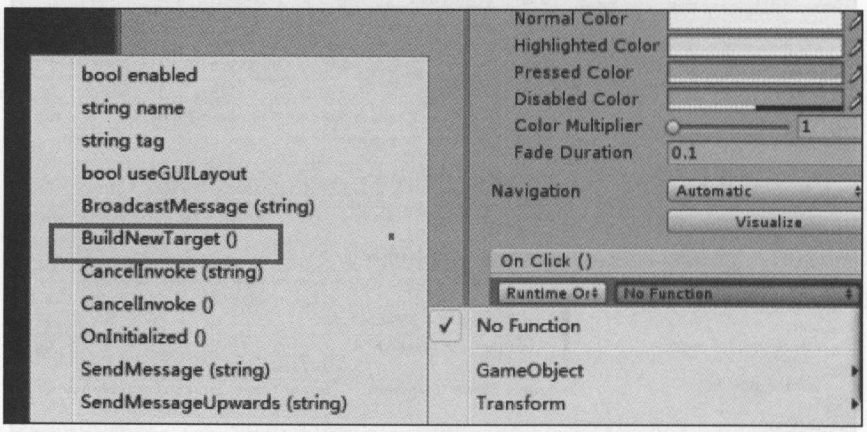
```
public void BuildNewTarget()
{
    mTargetBuildingBehaviour.BuildNewTarget("test", 50);
}
```

然后还需要返回 Unity 中进行一些细微的设置调整，将我们的脚本拖到 UserDefined TargetBuilder 下，并为 ImageTargetTemplate 进行赋值。



然后在 Button 上添加点击事件，步骤如下。





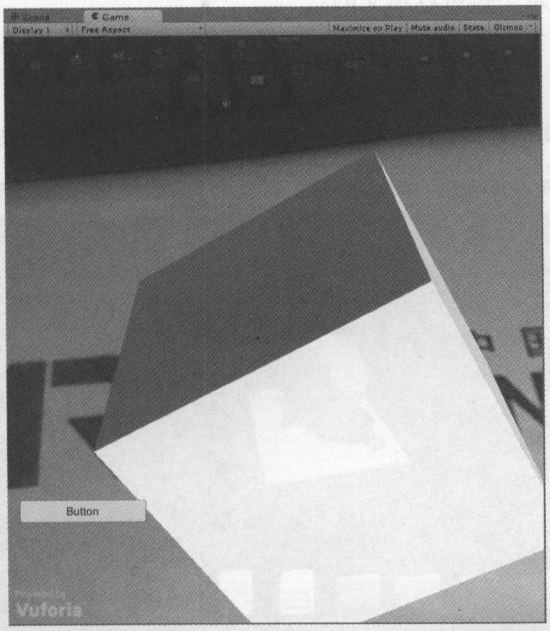
然后在 UserDefinedTargetBuilder 上勾选 Start scanning automatically，表示开启自动扫描。

5. 运行工程

首先拍摄一张图片，界面如下。



将我们需要自定义的识别图置于取景框内，点击 Button 便可识别出 Cube，将识别图移出摄像头取景范围，模型消失，再次扫描刚才的识别图，依然出现模型。



至此，Vuforia 的自定义目标识别介绍完毕，大家可以根据自己的需求发挥想象力，制作出属于自己的炫酷 AR 产品。

第3章 AR内容交互

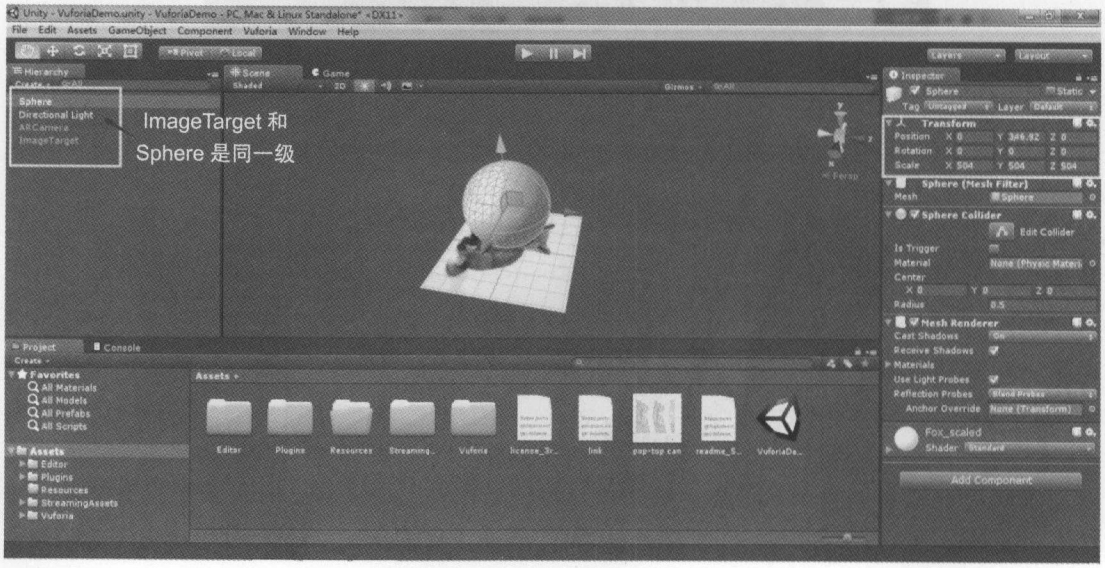
3.1 动态加载AR模型

在第2章中我们实现了通过Vufoira默认方式识别并显示三维模型。默认的加载模型是通过将3D物体直接放置在场景中并作为识别目标的子物体来实现的，这种实现方式的问题是，当场景中有很多识别目标后，需要一次性加载的模型内容会占用很大的内存。因此，在实际项目案例中需要动态加载识别目标对应的模型，在识别丢失之后删除模型，这样可以保证同一时刻内存的占用是手机可以承受的。

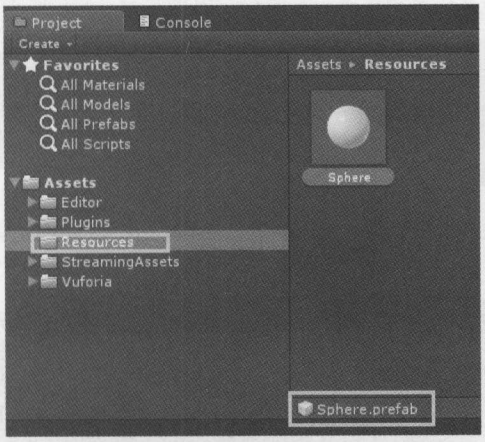
本节将通过预制件实例化的方式实现AR模型的动态加载和显示。

1. 创建预制件

将ImageTarget下调整好的模型拖到最上层，和ImageTarget在层级视图中保持平级关系。



在 Assets 目录中创建 Resources 文件夹，并将层级视图 Hierarchy 中的模型 Sphere 拖至 Resources 文件夹中，然后删除层级视图 Hierarchy 中的 Sphere 模型。



2. 为需要加载的预制件添加脚本

在工程视图 Project 中创建一个新的 C# 脚本，命名为 TrackObject.cs，该脚本是为了更方便地找到识别后显示的物体，将其拖到需要加载的预制件上。

在工程视图 Project 中找到 DefaultTrackableEventHandler.cs 这个脚本，按住 Ctrl+D 可以复制一份相同的脚本，选中新复制出来的脚本并按 F2 快捷键将脚本改名为 MyTrackable EventHandler.cs。然后双击打开，将类名修改为 MyTrackableEventHandler，找到监听识别成功和识别丢失的两个方法 OnTrackingFound 和 OnTrackingLost，默认代码如下：

```
//识别成功时调用的方法
    private void OnTrackingFound()
    {
        Renderer[] rendererComponents = GetComponentsInChildren<Renderer>(true);
        Collider[] colliderComponents = GetComponentsInChildren<Collider>(true);
        // Enable rendering:
        foreach (Renderer component in rendererComponents)
        {
            component.enabled = true;
        }
        // Enable colliders:
        foreach (Collider component in colliderComponents)
        {
            component.enabled = true;
        }
    Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " found");
}

//识别丢失时调用的方法
private void OnTrackingLost(){
    Renderer[] rendererComponents = GetComponentsInChildren<Renderer>(true);
    Collider[] colliderComponents = GetComponentsInChildren<Collider>(true);
```

```
        // Disable rendering:
        foreach (Renderer component in rendererComponents)
        {
            component.enabled = false;
        }
        // Disable colliders:
        foreach (Collider component in colliderComponents)
        {
            component.enabled = false;
        }
        Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " lost");
    }
```

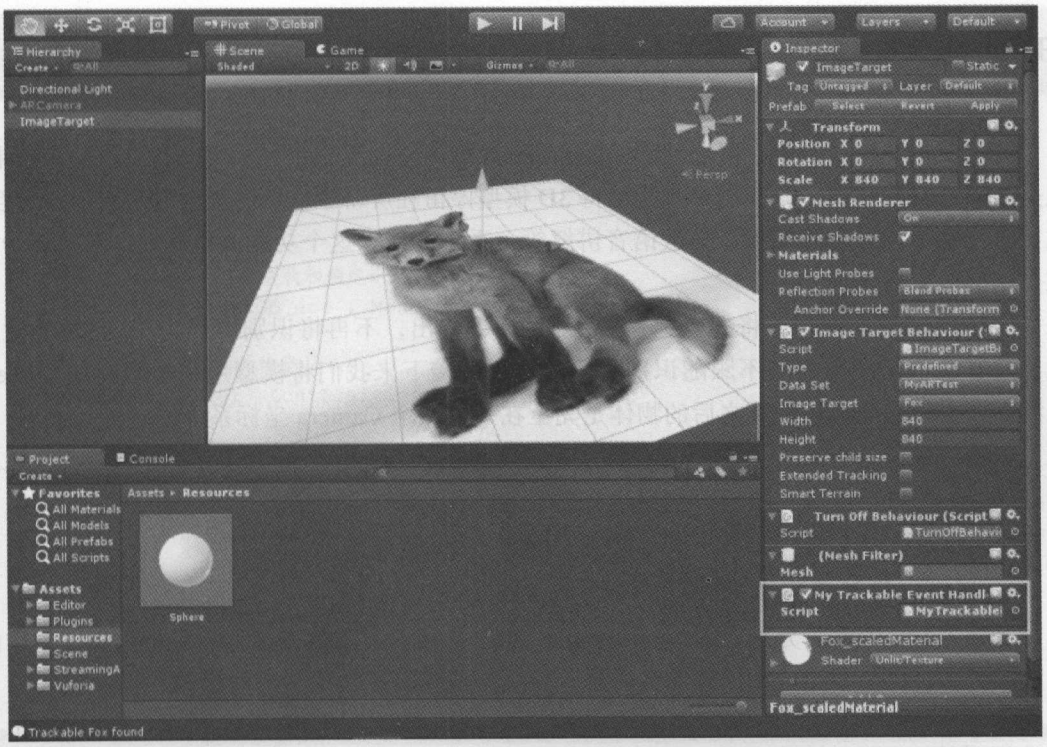
在默认的这两个方法实现中，OnTrackingFound 函数在识别图像成功之后被调用，OnTrackingLost 函数在识别丢失之后被调用。该脚本利用了查找当前识别目标所有子物体的渲染器 Renderer 组件，通过其打开或关闭来实现 3D 物体的显示和隐藏。现在我们通过动态加载的方式来显示 3D 模型，需要删除 OnTrackingLost 和 OnTrackingFound 两个函数中的内容，并将代码修改为动态加载的方式实现：

```
    //识别成功时调用的方法
    private void OnTrackingFound()
    {
        //动态加载模型并实例化到场景中
          GameObject obj = GameObject.Instantiate(Resources.Load("Sphere")) as
GameObject;
        //生成之后设定它的父级为识别图物体
        obj.transform.parent = this.transform;
        //设定物体与识别图在同一位置，这里可以根据需求设定
        obj.transform.position = this.transform.position;
    }

    //识别丢失时调用的方法
    private void OnTrackingLost()
    {
        //从子物体中查找所有 TrackObject 组件
        TrackObject to=GetComponentInChildren<TrackObject>();
        //如果找到显示物体，则销毁物体并释放无用资源
        if (to != null)
        {
            Destroy(to.gameObject);
            Resources.UnloadUnusedAssets();
        }
    }
```

3. 效果实现

选择 ImageTarget，在 Inspector 面板上选择 Remove Component，并将 DefaultTrackable EventHandler 脚本移除，同时把 MyTrackableEventHandler 脚本添加到 ImageTarget 上。



最后，点击运行以测试最终效果，如下图所示。



以上为动态加载 AR 中 3D 物体的步骤和方法。

3.2 模型脱卡功能的实现

在常见的 AR 项目中，识别到图像后会将 3D 物体叠加到识别目标之上并具有追踪效果，但是当识别目标丢失后，我们希望能够将 3D 模型停留在屏幕中心，并能够和用户进行交互，比如点击模型伴随动画切换、播放语音讲解等一系列功能。这个功能就是我们本节将要为大家介绍的脱卡功能。

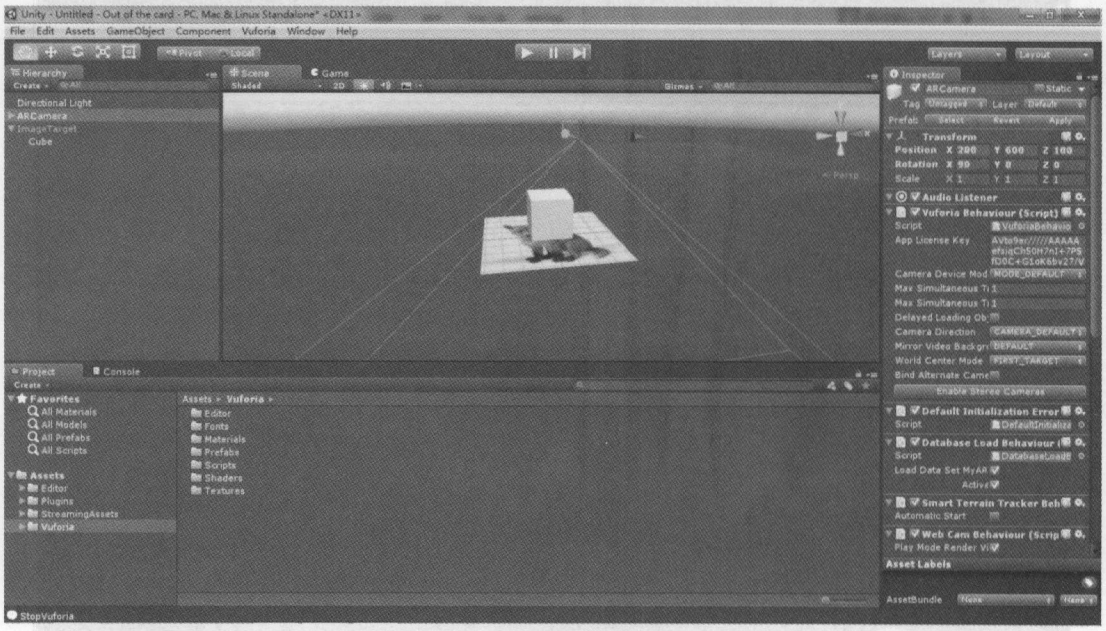
脱卡实现的原理是，将 3D 物体从识别目标下移出，不再将识别图作为 3D 物体的父对象，这样就能够实现模型不跟随识别物体的效果。接下来我们将模型用一个新建的 Camera 来渲染，也就是脱离识别图之后的物体是完全在另外一个 Camera 坐标系下的。

其实，脱卡功能还有更简单的实现方式，比如在 ARCamera 下创建空物体作为脱离之后模型的目标位置，然后在脱离识别图后将模型移动到空物体下。

在本节中，我们使用另外一种思路来实现脱卡功能。

1. 创建工程并设置参数

新建 Unity 工程，分别导入 Vuforia SDK 插件包和识别图数据包，然后根据第 1 章中的设置方法，将需要添加的物体添加到场景，并调整参数以正常识别并显示 3D 物体。



选择 ImageTarget，将其自身的 DefaultTrackableEventHandler 移除，并将该脚本复制一份命名为 MyTrackableEventHandler，然后附加到 ImageTarget 上。然后创建 TrackObject.cs 和 Rotate.cs 两个文件。

双击打开 Rotate.cs 文件，并在该脚本 Update() 方法中添加一行代码以使物体旋转，代码如下：

```
//Rotate.cs
// 刷新函数，每一帧被调用一次
void Update ()
{
    // 让物体绕 Y 轴自转
    transform.Rotate(new Vector3(0f,1f,0f));
}

//TrackObject.cs
//TrackObject 类用来查找和确定识别后显示的物体
public class TrackObject : MonoBehaviour
{
    // 脱卡之后模型的位置
    public Vector3 LostPosition;
}
```

2. 脚本修改

修改 MyTrackableEventHandler.cs 中的 OnTrackingFound 和 OnTrackingLost 两个方法。

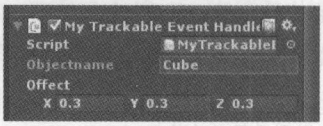
```
// 识别成功时调用的方法
private void OnTrackingFound()
{
    // 获取所有 TrackObject 物体并将它们都销毁
    TrackObject[] objects = FindObjectsOfType<TrackObject>();
    foreach(TrackObject to in objects)
    {
        Destroy(to.gameObject);
    }
    // 卸载无用资源
    Resources.UnloadUnusedAssets();
    // 加载需要显示的模型并实例化到场景中
    GameObject obj= GameObject.Instantiate(Resources.Load(objectName))as GameObject;
    // 将父物体设置为 ImageTarget
    obj.transform.parent = this.transform;
    // 设置物体位置
    obj.transform.position = this.transform.position;
}

// 识别目标丢失时调用
private void OnTrackingLost()
{
    // 从子物体中获取所有 TrackObject 组件
    TrackObject to = GetComponentInChildren<TrackObject>();

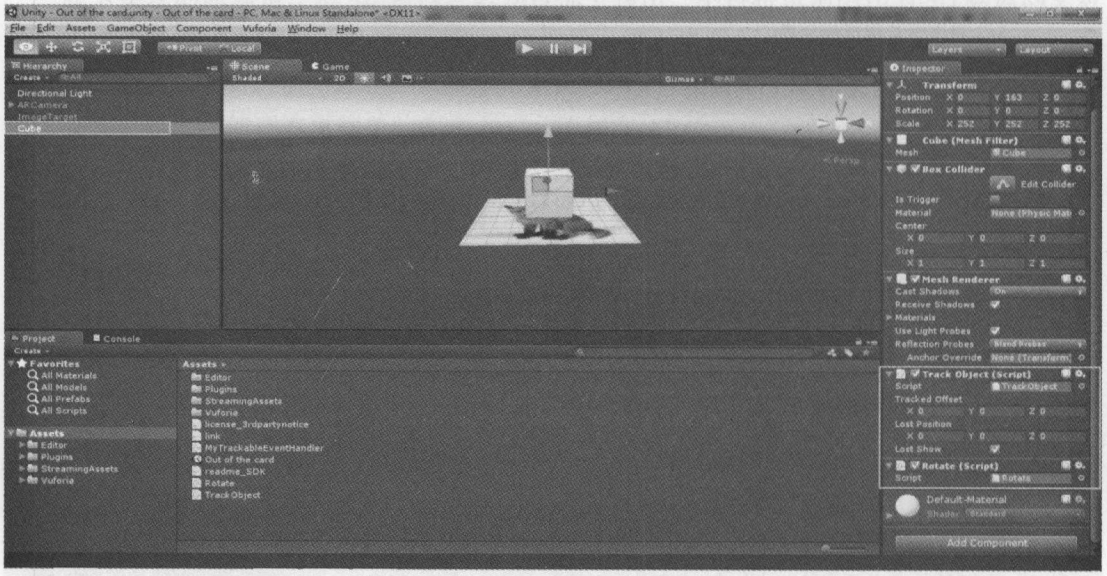
    if (to != null)
    {
        // 将 3D 物体的父对象从识别目标下移出
            to.transform.parent = this.transform.parent;
        // 设置物体的 layer 值，这里用另一个 Camera 来渲染脱卡之后的模型
            to.gameObject.layer = 10;
        // 将物体的位置设置为 TrackObject 脚本中设置好的位置
            to.transform.position = to.LostPosition;
    }
}
```

3. 场景搭建

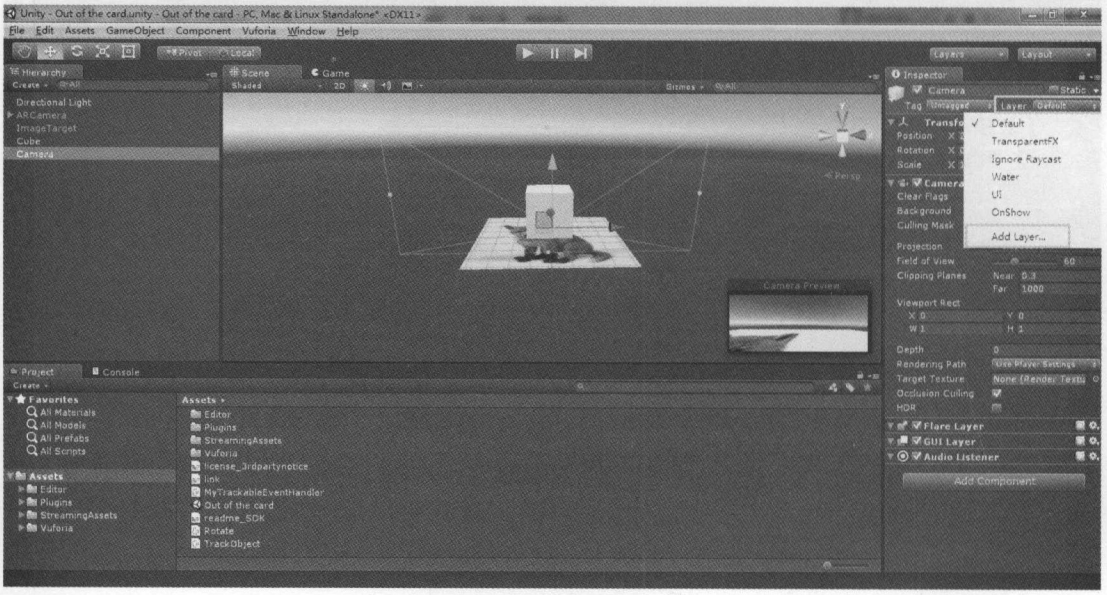
选择 ImageTarget，在 Inspector 面板中找到 MyTrackableEventHandler 脚本，设定以下参数。



然后将 Cube 拖到与 ImageTarget 平级的层级，如下图所示。

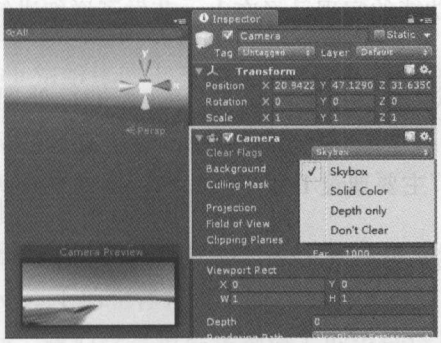


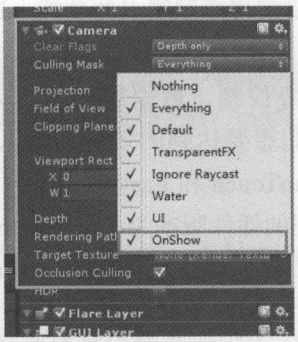
接下来在 Hierarchy 面板中右击创建一个新的 Camera，选择其 Inspector 面板 Layer 标签中的 Add Layer 命令创建一个新的层。



在 Layer 10 的位置创建一个新的图层 OnShow。再调整刚才新建的 Camera 的 Clear Flags 属性如下。

选择 Depth only，然后在 Culling Mask 下选择 OnShow 层，如下图所示。

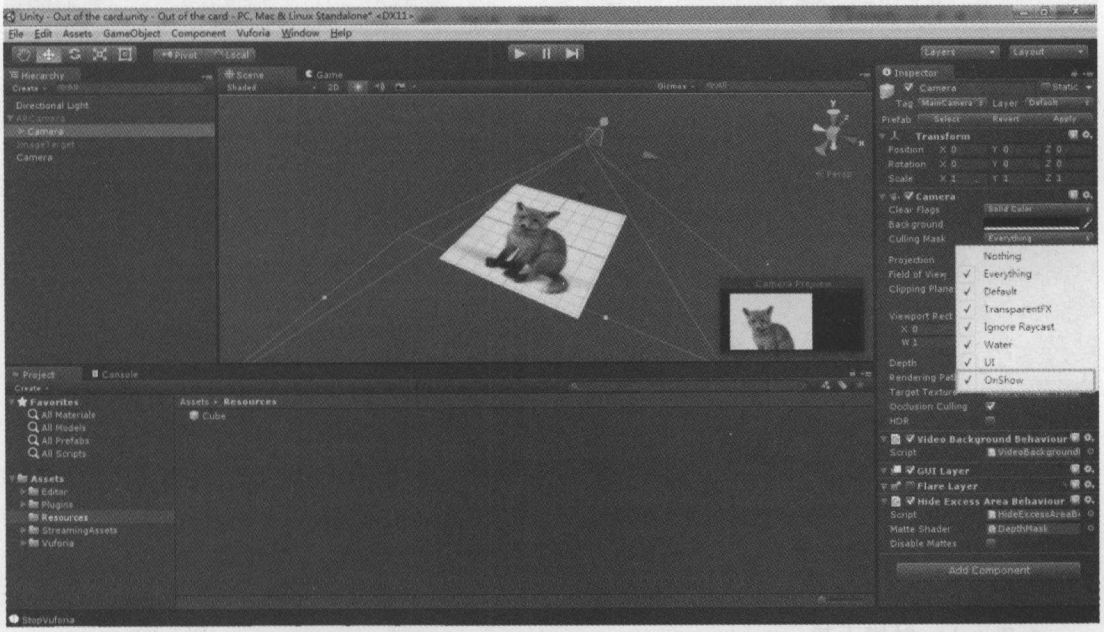




4. 效果实现

在 Assets 文件夹中新建一个名为 Resources 的文件夹，将 Cube 拖到该文件夹中生成 prefab 预制件，并删除在层级视图 Hierarchy 中的 Cube 物体。

选中 ARCamera 子物体下的 Camera 对象，在其 Inspector 面板中找到 Culling Mask 属性并取消勾选 OnShow。



接下来，点击运行便可测试脱卡效果的实现。

3.3 手势控制

目前市场上流行的 AR 产品以儿童教育类产品居多，这些产品中大多会有手势交互的功

能，本节就将讲解如何在 AR 产品中加入手势交互的功能。设想以下场景，在识别一个动物的卡片后，会出现该动物的 3D 立体影像，在识别丢失后，动物模型会停留在屏幕中心，并可以通过手势控制其旋转和缩放。

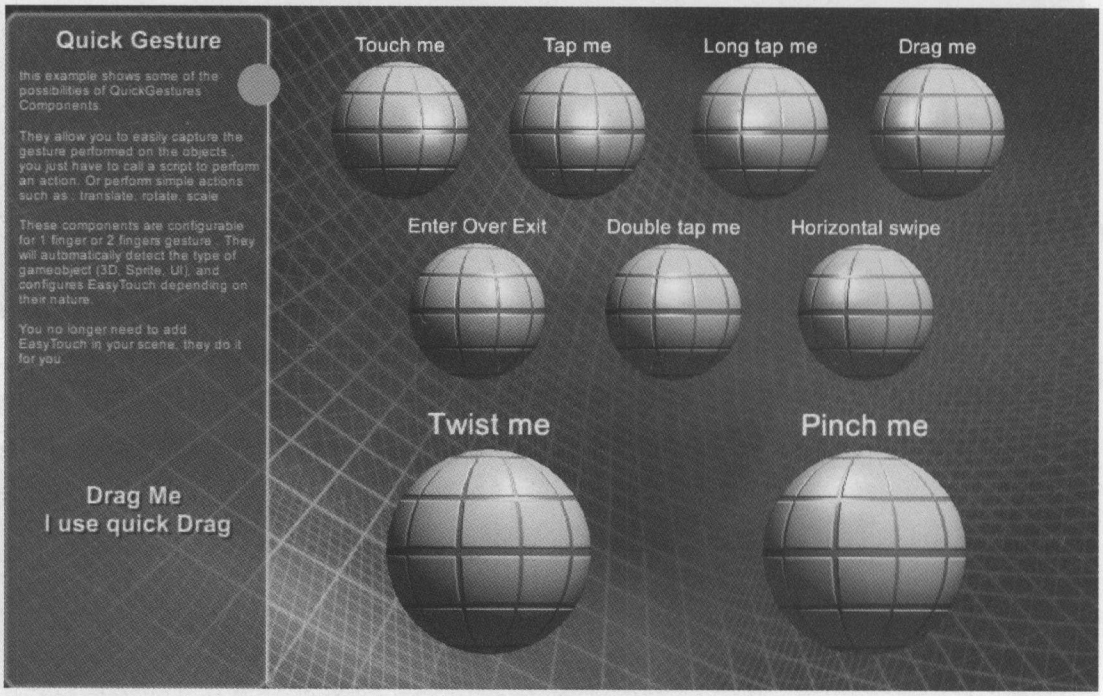
我们需要实现模型脱卡的功能，这个功能的实现已经在上一节做了详细讲解，本节不再赘述。

关于手势控制，最原始的方式是通过 Unity 3D 自带的 Input 类进行处理，但是为了快速实现我们想要达到的效果，可以借助一些插件来实现。用于手势处理的插件有 Finger Gesture、LeanTouch 和 EasyTouch 等，本节主要通过 EasyTouch 这个插件来讲解手势控制功能（本节使用的插件版本为 EasyTouch 5）。

接下来，我们就逐步讲解 EasyTouch 的使用方法。

1. 查看 EasyTouch 案例场景

正确导入 EasyTouch 插件之后，在 Project 视图中找到 QuickGesture 案例场景，这个场景中展示了插件中大部分手势的案例，该场景打开后的界面如下图所示。



上图中的场景展示了下列手势：

- Touch：触摸手势。
- Tap：点击手势。
- Long tap：长按手势。
- Drag：拖动手势。
- Enter Over Exit：滑过手势。
- Double tap：双击手势。

- Swipe：滑动手势。
- Twist：扭转手势。
- Pinch：两个或多个手指挤压，常用于缩放。

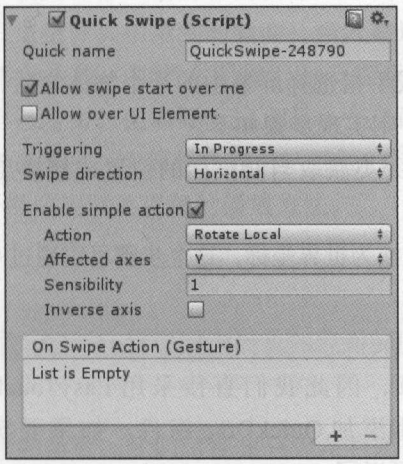
2.EasyTouch 初始化

EasyTouch 初始化的方式非常简单，只需在场景中创建一个空的 GameObject，将其命名为 EasyTouch，然后在 Inspector 面板上点击 AddComponent 添加 EasyTouch.cs 脚本文件即可。

3. 旋转模型

初始化模型的工作完成之后，接下来演示旋转模型的实现方式。查看之前案例场景中 Horizontal Swipe 下的物体，你会发现当鼠标在该物体上左右滑动时，就可以让物体发生水平方向的旋转，我们查看被旋转物体上的组件信息，发现该物体拥有一个名为 QuickSwipe 的组件。

因此，只要给需要旋转的物体添加 QuickSwipe 组件，即可让物体根据手势滑动的方向进行旋转。Quick Swipe 组件的参数如下图所示。



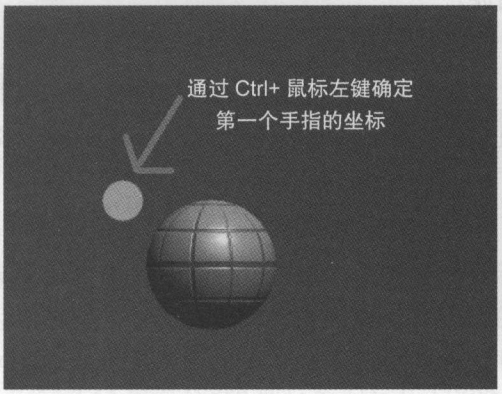
上图中的 Swipe direction 选项表示判断手势滑动的方向，例如 Horizontal 表示水平方向，Vertical 表示竖直方向。Action 中的选项表示当手势触发之后，所影响的物体会发生哪些类型的动作，可以选择的动作有 Rotate（旋转）、Translate（移动）和 Scale（缩放），在此我们选择 Rotate 选项即可。Affected axes 表示被影响的坐标轴，如果是水平方向渲染，则选择 Y 轴。Sensibility 表示灵敏度，可以根据灵敏度调整旋转的速度，Inverse axis 表示反向。

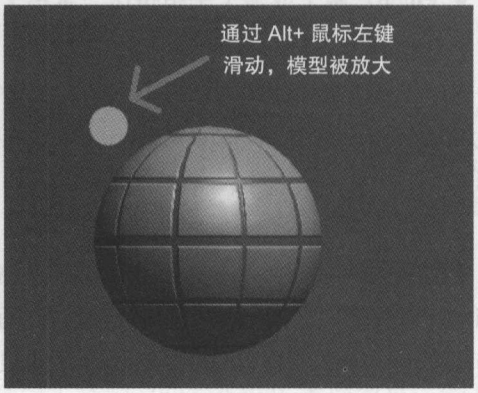
4. 缩放模型

在手机 App 中，一般是通过两个手指在触摸屏幕上的距离远近变化来控制模型的大小，因此我们采用 EasyTouch 中的另外一种 Pinch 事件。在上述案例场景中我们可以看到这个事件，但是在 Unity 编辑器中无法通过鼠标模拟多点触控，EasyTouch 的解决方案是：首先用 Ctrl+ 鼠标左键确定第一个手指的位置，然后通过 Alt+ 鼠标左键来控制第二个手指的位置与第一个手指的位置之间的距离。

（1）Ctrl + 鼠标左键确定第一个手指的位置。

（2）Alt + 鼠标左键滑动控制模型缩放。





在实现缩放控制的时候，EasyTouch 为我们提供的接口也特别简单。与旋转模型类似，只需要给被旋转的物体加上一个 QuickPinch 组件，但是不要忘记给物体再添加一个碰撞器组件。

5. 拖动模型

拖动模型是指，当鼠标或手指选择屏幕中的某个物体之后，拖动手指，物体会跟随手指进行移动，如果不借助插件，则实现步骤如下。

（1）通过 Unity 中的 Input 类获取当前鼠标的位置或触摸点的位置，返回的坐标信息为屏幕坐标。

（2）将获取的屏幕坐标转换为世界坐标，这个步骤可以通过 Camera 中的 ScreenToWorldPoint 方法来实现。

（3）将模型的位置移动到转换后的世界坐标位置。

上述实现过程比较麻烦，因此我们直接采用 EasyTouch 中的 QuickDrag 组件实现。我们只需要给被拖动的物体添加 QuickDrag 组件。如果要实现两个手指拖动，则添加 TwoDragMe 这个脚本组件就可以了。

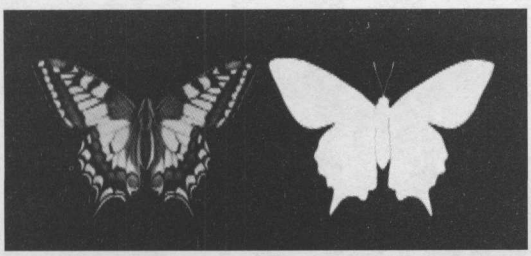
本节关于手势控制的内容已经介绍完毕，其实 EasyTouch 还拥有其他一些手势控制和功能，大家可以参考它的帮助文档。

第4章 AR互动大屏：透明视频的实现

互动大屏是AR中常见的应用方式，在互动大屏的开发过程中，常常会用到透明视频的播放。比如，在商场中放置一块大屏幕，平时看上去像一面镜子，实际上屏幕背后的主机连接了摄像头，屏幕上显示的只是摄像头拍摄到的画面，当人靠近屏幕或者摄像头捕捉到人的图像或动作时，就会生成虚拟的人物，如美女明星。对于观众来说，当他看向屏幕的时候，会以为美女明星真的出现在自己身旁，但实际上这只是一个美丽的错觉而已。本章主要介绍如何在Unity中实现透明视频的播放，这部分内容包括视频的制作、特殊Shader的编写、在Unity中不借助AR SDK打开摄像头以及视频的播放方法。

4.1 使用AE制作带Alpha通道的视频

为了更好地实现透明视频的播放，我们将视频分为左右两部分，左边为正常的视频，右边只有该视频的黑白剪影。通过Shader获取右边位置的颜色信息，如果右边对应坐标的颜色值为黑色，则左边视频相应位置的Alpha值为0（透明），否则保留原本的透明度即可。借助下面的图像会更容易理解这一原理。



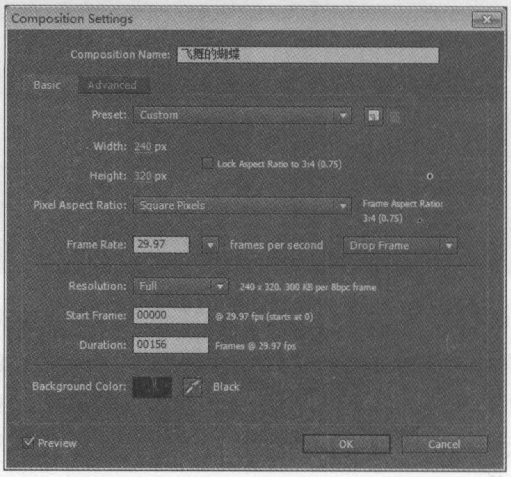
上图中左边的视频会根据右边的剪影进行透明度的设置，最终显示的视频内容只有左边的蝴蝶图案。

接下来我们介绍如何用 AE（Adobe After Effects）软件来处理视频。

（1）准备需要进行通道处理的视频素材，本节使用视频素材名为“飞舞的蝴蝶”。

（2）打开 AE，选择 File_import_file（快捷键 Ctrl+I）找到相应的素材文件，点击打开。

（3）新建合成（Ctrl+N），命名为“飞舞的蝴蝶”，尺寸改为 240px × 320px，素材时长 156 帧（可以查看素材的时间长度），点击 OK 按钮。



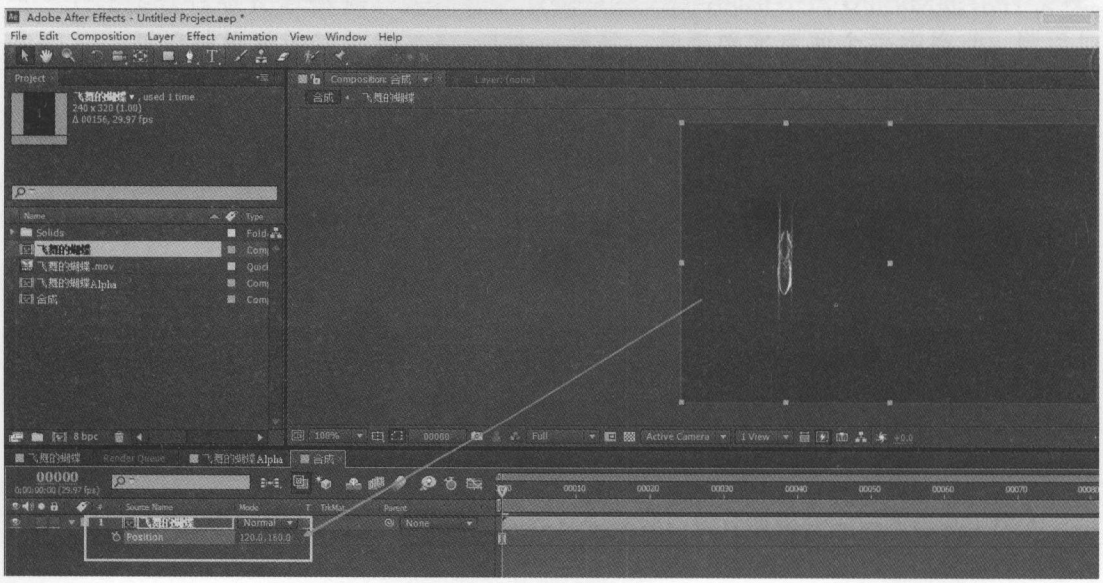
（4）将素材拖进合成里面，按 S 键进行素材大小的调整，让画面居中显示，并且保证画面不会被裁切掉。

（5）复制（Ctrl+D）“飞舞的蝴蝶”合成，命名为“飞舞的蝴蝶 Alpha”。

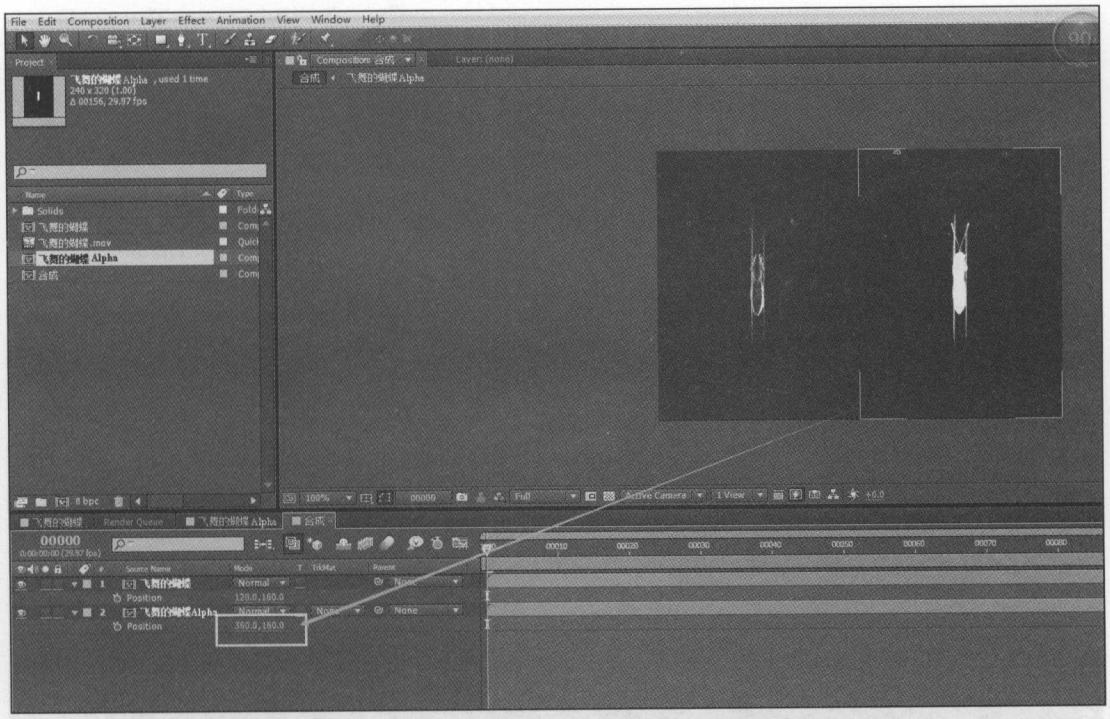
（6）在合成中右键创建一个白色的固态层（Ctrl+Y），此层必须在素材的下面，并选择 Alpha 模式为 Alpha Matte “飞舞的蝴蝶”.mov。

（7）新建合成，大小为 480px × 320px，目的是将视频宽度放大为原视频的两倍。

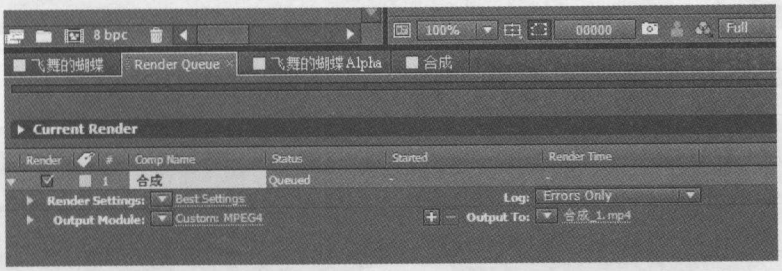
（8）将之前的“飞舞的蝴蝶”合成放入到新“合成”中，按 P 键移动为 240 的一半（120）。



（9）将之前的“飞舞的蝴蝶 Alpha”合成放入新“合成”中，按 P 键移动为 360。



（10）选择合成（Ctrl+M）输出一个 .mp4 格式的文件。



以上步骤可将一个普通视频处理成在程序中用 Shader 进行透明处理的视频，下一节我们将介绍如何通过 Unity 中的 Shader 显示不透明区域的视频画面。

4.2 编写可播放透明视频的 Shader

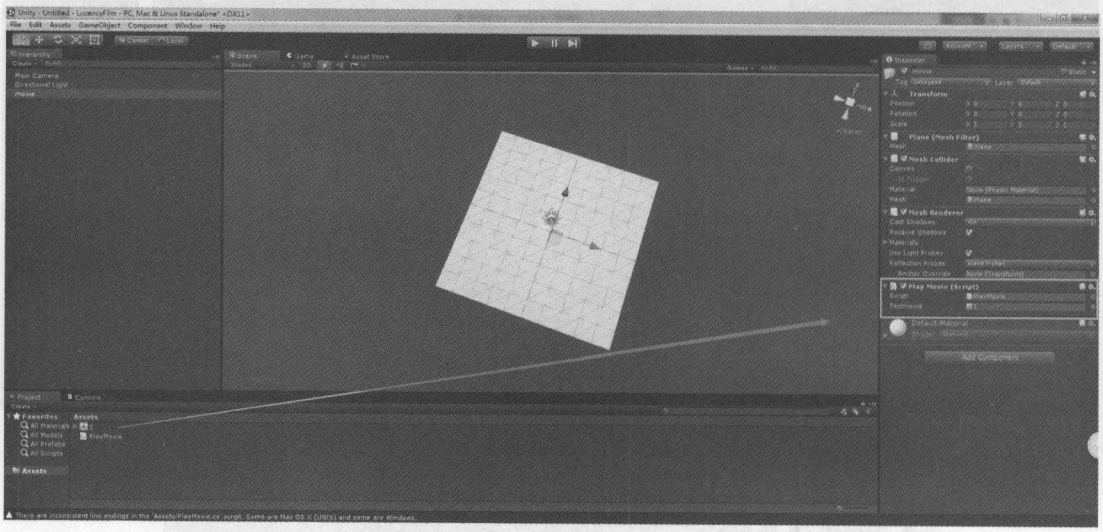
本节的主要内容是编写一个着色器（Shader），用于将上一节中的视频按照我们需要的方式进行透明度（Alpha）处理。

新建 C# 脚本，命名为 PlayMovie 并写入如下代码：

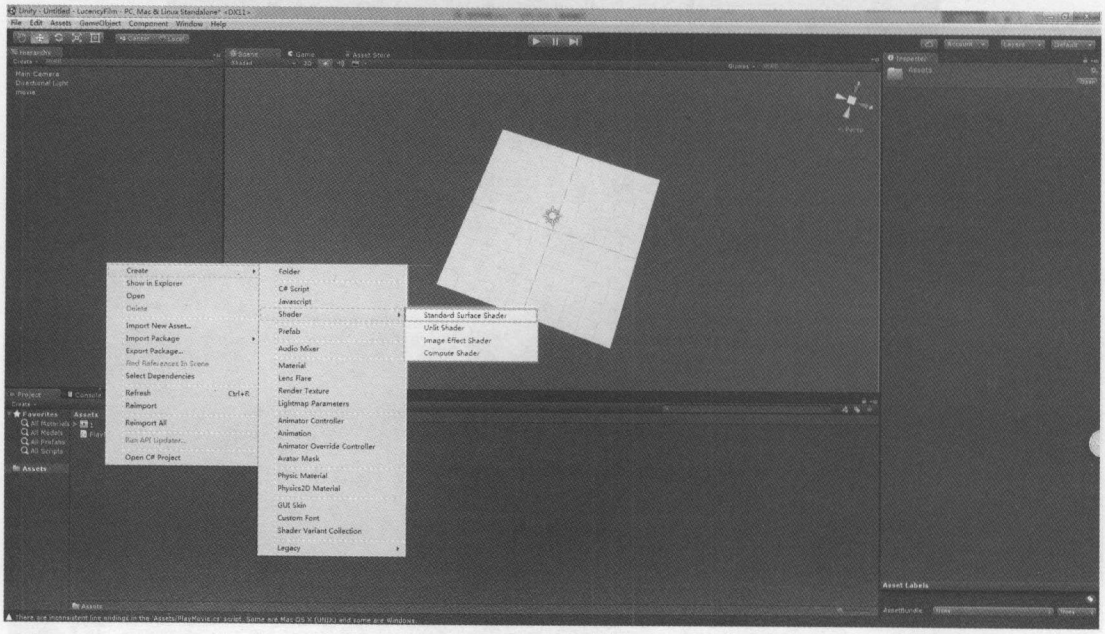
```
// 声明一个公共的 MovieTexture 变量 testmovie，这是一个 .ovg 格式的影片
```

```
public MovieTexture testmovie;
void Start()
{
    //这行代码是用来测试的，之后需要将其删掉
    testmovie.Play();
    //设置影片是可以循环的，如果不想循环，则设置为 false
    testmovie.loop = true;
}
```

在场景中新建一个面片（Plane），更名为 movie，将 PlayMovie 脚本拖上去，同时在 MovieTextrue 处将视频素材拖上去。



接下来创建一个 Shader，命名为 movieshader。



将Shader中原来的代码都删除，然后再写入如下代码。

```
Shader "Custom/shaderMovie"
{
    Properties
    {
        // 声明一个2D贴图，默认的颜色为黑色 _MainTex("Color (RGB)", 2D) = "black"
    }

    //SubShader，渲染方案。每个可编程Shader中至少有一个SubShader，可为不同显卡编写不同SubShader
    SubShader{

    // 子着色器使用标签（Tags）来告诉渲染引擎何时和如何渲染对象
    //Tags { "标签1" = "值1"  "标签2" = "值2" }
    // 标签是标准的键值对，可以有任意多个，常用的标签如下
    // Queue——队列标签，用来决定对象被渲染的次序
    // 着色器决定对象所归属的渲染队列，任何透明物体都可以通过这种方法确保在不透明物体渲染之后渲染

    //ShaderLab中有4种预定义的渲染队列
    //Background——后台，这个渲染队列在所有队列之前被渲染，用于渲染天空盒之类的对象
    //Geometry——几何体，这个队列是默认队列，用于大多数对象。不透明的几何体使用这个队列
    //Transparent——透明，这个渲染队列在几何体队列之后被渲染，采用由后到前的次序。任何采用Alpha混合的对象（不对深度缓冲产生写操作的着色器）都应该在这里渲染（如玻璃、粒子效果）
    //Overlay——覆盖，这个渲染队列用于实现叠加效果。任何需要最后渲染的对象都应该放置在此处（如镜头光晕）

    Tags{ "Queue" = "Transparent" "RenderType" = "Transparent" }

    //CG代码必须用

    //CGPROGRAM

    //...

    //ENDCG括起来

    //pragma surface 是定义unity Surface Shader中的函数定义
    CGPROGRAM
    #pragma surface surf NoLighting alpha

    //Alpha的数值决定了一张贴图的透明度
    fixed4 LightingNoLighting(SurfaceOutput s, fixed3 lightDir, fixed atten) {
            fixed4 c;
            c.rgb = s.Albedo;
            c.a = s.Alpha;
            return c;
    }

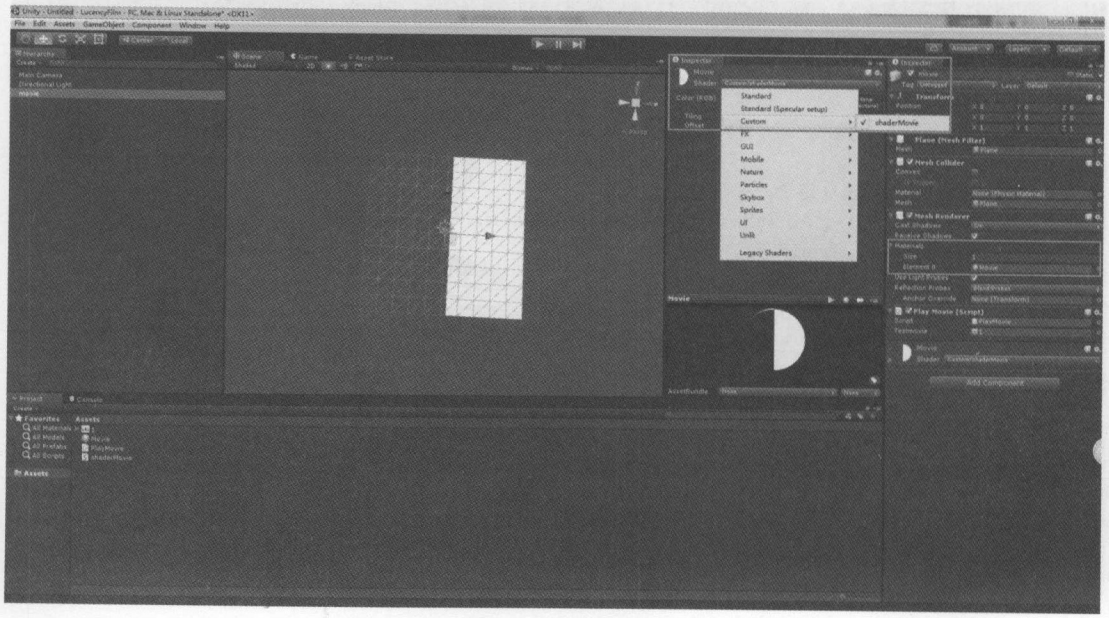
    // 声明了一个Input的结构体
    struct Input
    {
            float2 uv_MainTex;
    };
```

```
        sampler2D _MainTex;

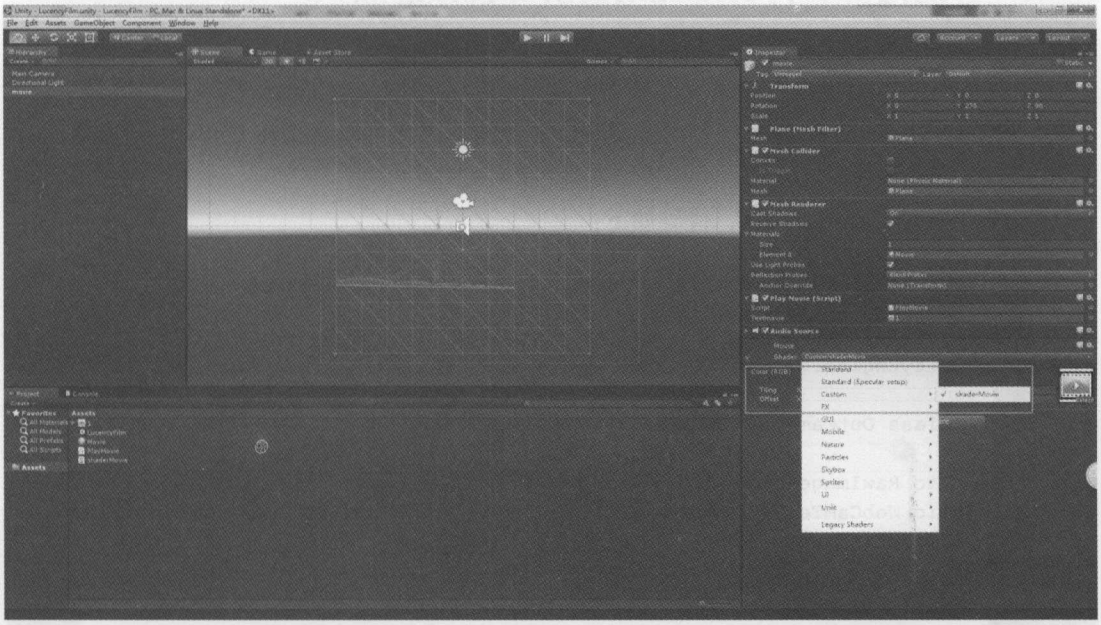
        void surf(Input IN, inout SurfaceOutput o)
            {
                o.Emission = tex2D(_MainTex, IN.uv_MainTex).rgb;
                // 将黑白中的一部分 Alpha 值设为 0
            // 这个阈值及大于、小于号需要根据具体的电影文件来设置，一般在 0.45 ～ 0.5 之间
            if(IN.uv_MainTex.x >= 0.5)
                {
                    o.Alpha=0;
                }
                // 将另一半画面的 Alpha 值设置为黑白部分的 RGB 值
                else
                {
                    o.Alpha = tex2D(_MainTex, float2(IN.uv_MainTex.x+0.5, IN.uv_MainTex.
y)).rgb;
                }
            }
            ENDCG
        }
    }
```

显示屏从左到右依次是数字 0 和 1，上述代码将画面分割为两半，然后再改变各自的透明通道（Alpha）的值。

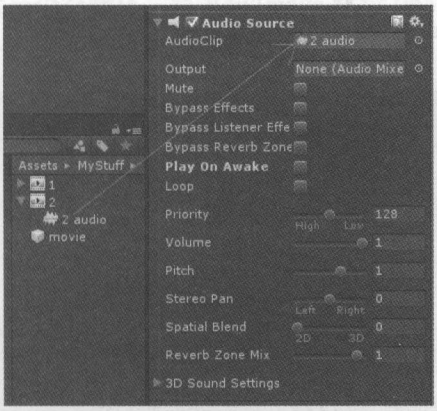
接下来创建一个材质球，命名为 movie，将 shadermovie 赋给材质球 movie，然后把材质球拖给面片 Plane，效果如下。



把视频拖给材质球，然后调整到合适属性，出现如下效果。



如果视频有声音，就为它添加 AudioSource 的组件，并把 .ogv 格式下的 audio 文件拖至 AudioClip。



最后，点击运行以测试效果，可以看到最终游戏界面上显示的图像只有蝴蝶的图案，达到了我们预期的透明效果，如下图所示。



4.3 开启摄像头并实现互动大屏的视频播放

首先需要开启摄像头，如果我们采用 Vuforia SDK 中的 ARCamera 实现开启摄像头，会发现早期的 Vuforia SDK 是不支持导出到 PC 平台的，因此可以直接在 Unity 中开启摄像头。具体的实现步骤如下。

新建一个脚本 OutCamera，代码如下：

```
using UnityEngine;
using System.Collections;
using System.Threading;
using UnityEngine.UI;

public class OutCamera : MonoBehaviour
{
    public RawImage image;
    public WebCamTexture webTex;
    public string deviceName;

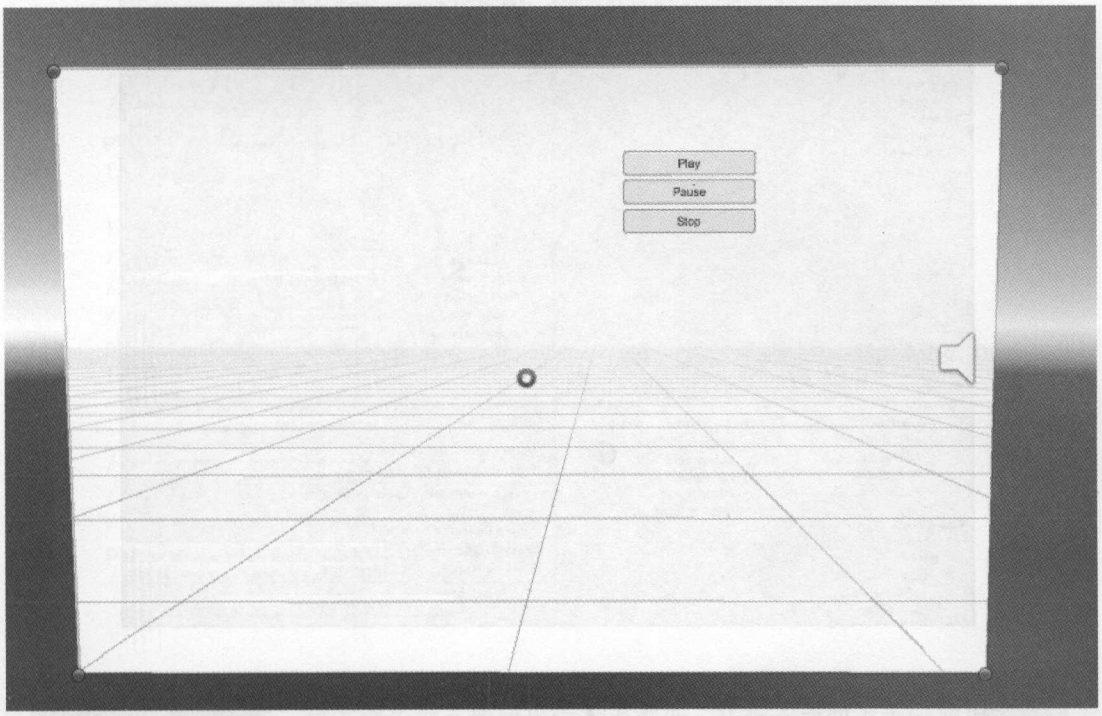
    void Start ()
    {
        //调用启动的协程，开启摄像头
        StartCoroutine("CallCamera");
    }

    private void Update()
    {
        if (webTex != null)
        {
            image.texture = webTex;
        }
    }

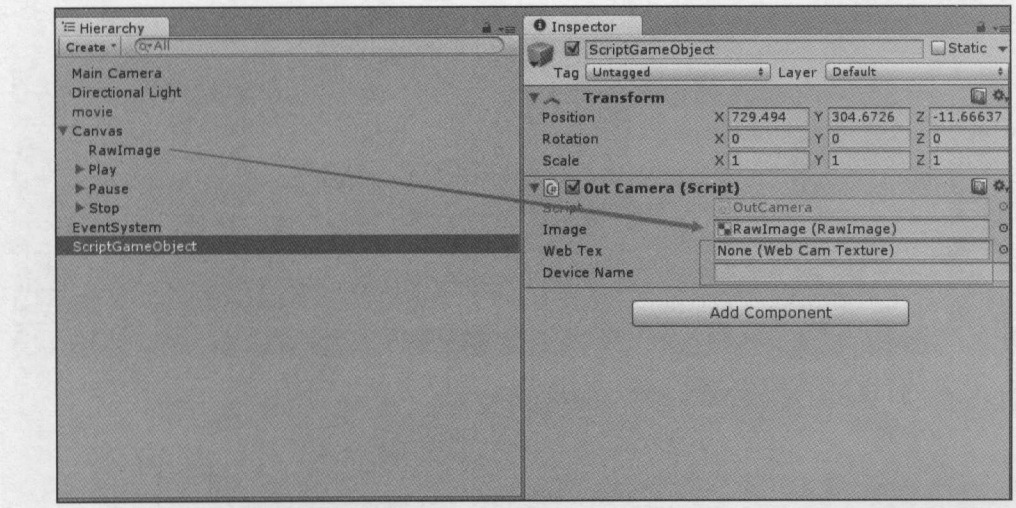
    private IEnumerator CallCamera()
    {
        yield return Application.RequestUserAuthorization(UserAuthorization.WebCam);
        if (Application.HasUserAuthorization(UserAuthorization.WebCam))
        {
            WebCamDevice[] devices = WebCamTexture.devices;
            deviceName = devices[0].name;
            //设置摄像机摄像的区域
            webTex = new WebCamTexture(deviceName,Screen.width,Screen.height,20);
            //如果需要导出到移动设备上，需要使用下面的代码
            //webTex = new WebCamTexture(deviceName, 800, 600, 20);
            //开始摄像
            webTex.Play();
        }
    }
}
```

创建一个 RawImage 以及 Play、Pause、Stop 三个按钮，将 RawImage 调整为全屏显示，

并将三个按钮的位置调整好，如下图所示。

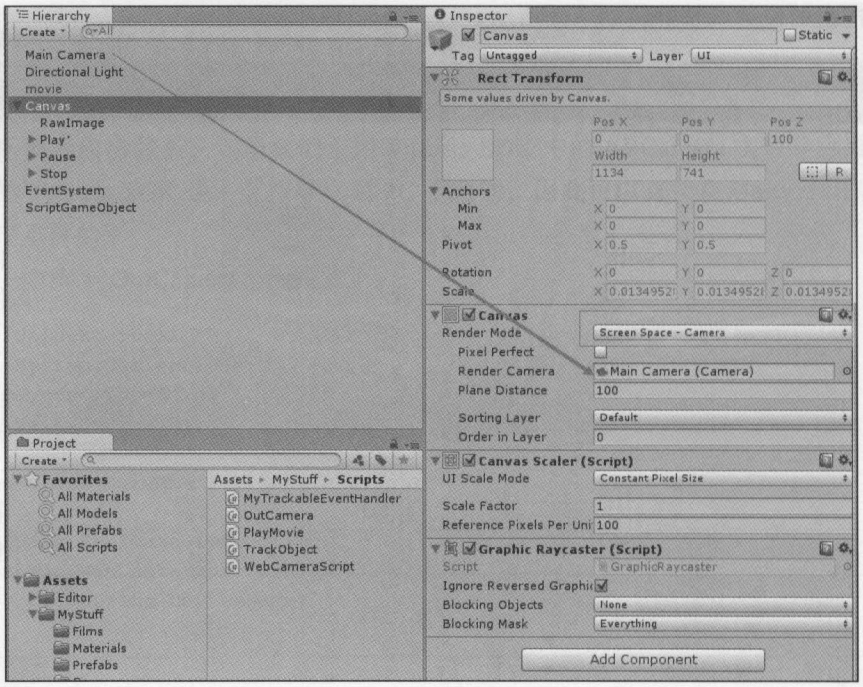


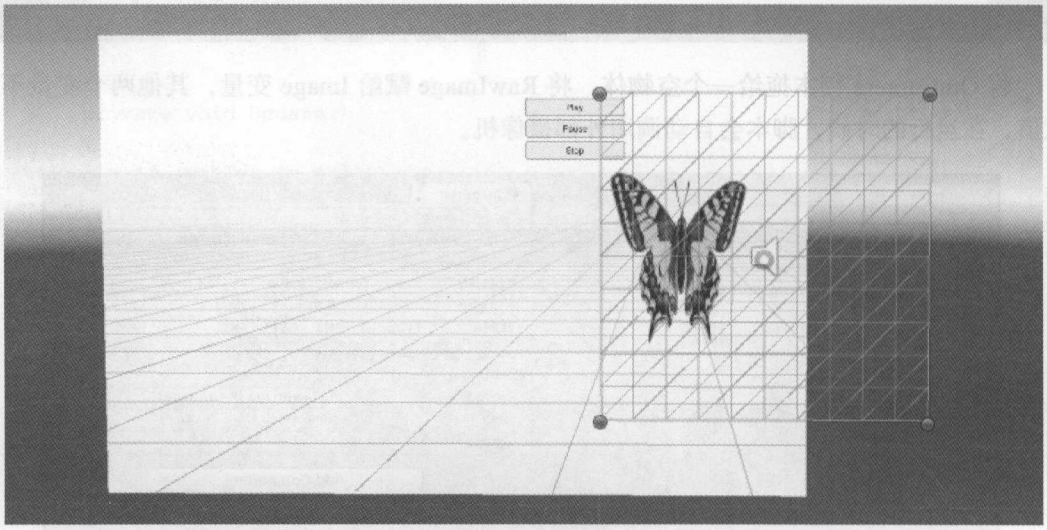
将 OutCamera 脚本拖给一个空物体，将 RawImage 赋给 Image 变量，其他两个变量不用赋值，在运行的时候，脚本会自动调用外部摄像机。



将 Canvas 的 Render Mode 属性设置为 Screen Space-Camera，并且将 Render Camera 设置为 Main Camera。

将 Main Camera 的投影方式（Projection）设置为正交投影（Orthographic）。然后调整之前做好的 movie 以使其处于合适的位置。





打开之前的 PlayMovie 脚本，改变其中的代码如下：

```
using UnityEngine;
using System.Collections;

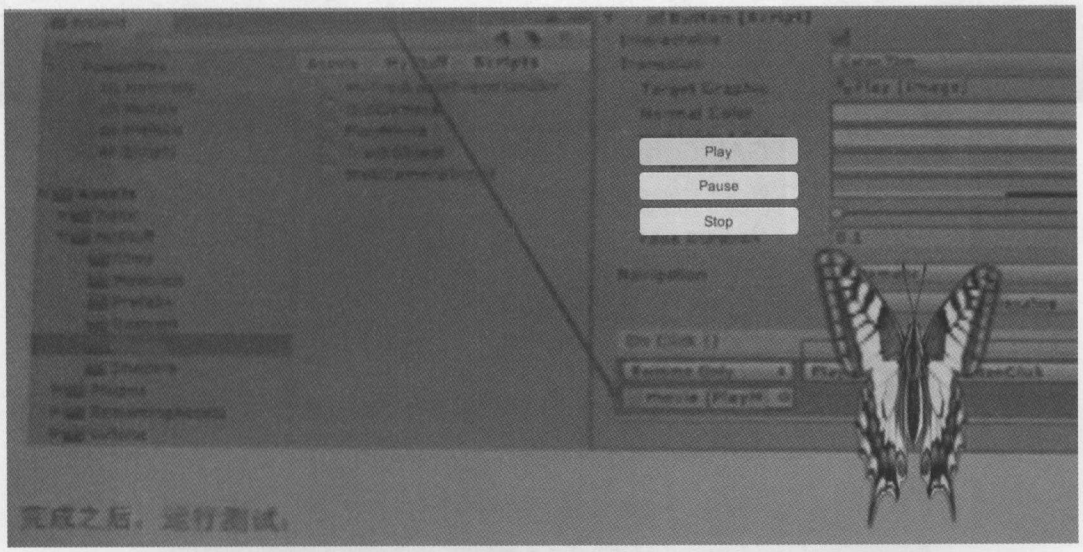
public class PlayMovie : MonoBehaviour
{
    public MovieTexture testmovie;

    void Start()
    {
```

```
        testmovie.loop = true;
    }

    /// <summary>
    /// Play按钮点击事件
    /// </summary>
    public void OnPlayButtonClick()
    {
        testmovie.Play();
    }
    /// <summary>
    /// Pause按钮点击事件
    /// </summary>
    public void OnPauseButtonClick()
    {
        testmovie.Pause();
    }
    /// <summary>
    /// Stop按钮点击事件
    /// </summary>
    public void OnStopButtonClick()
    {
        testmovie.Stop();
    }
}
```

脚本添加完成之后，接下来为每个按钮指定点击事件，让每个按钮点击时调用正确的方法。完成之后，运行测试。



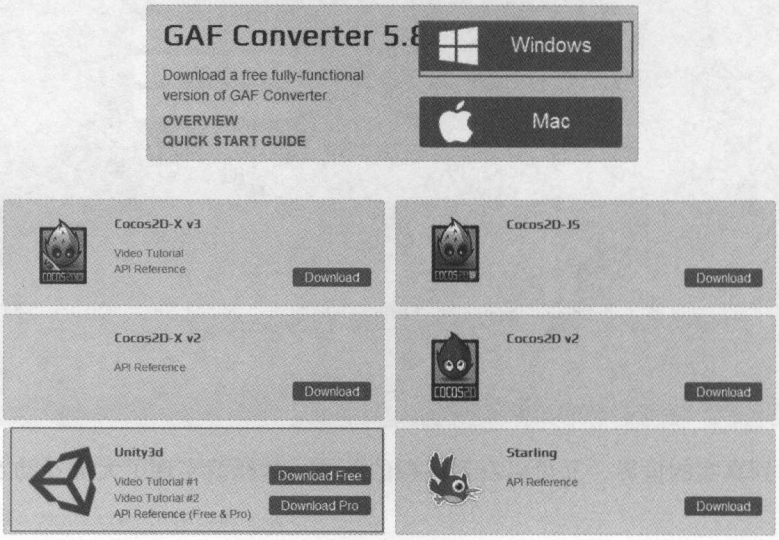
既可以看到真实的世界，又可以看到虚拟的影像，这样就实现了大屏互动！

第5章 AR动画：AR与Flash动画的结合

本章的主要内容是将 Flash 动画导入 Unity，实现 AR 动画的效果。Unity 3D 本身是不支持播放 Flash 动画的，所幸我们找到了一款插件，可以直接将 Flash 动画导出并在 Unity 环境下播放，这个插件叫 GAF Converter，使用方便简洁，能够快速实现我们想要达到的效果。

5.1 准备 GAF 插件

登录 GAF 官网 http://gafmedia.com/downloads，分别下载电脑版和 Unity 版，本章下载的版本为 5.4。

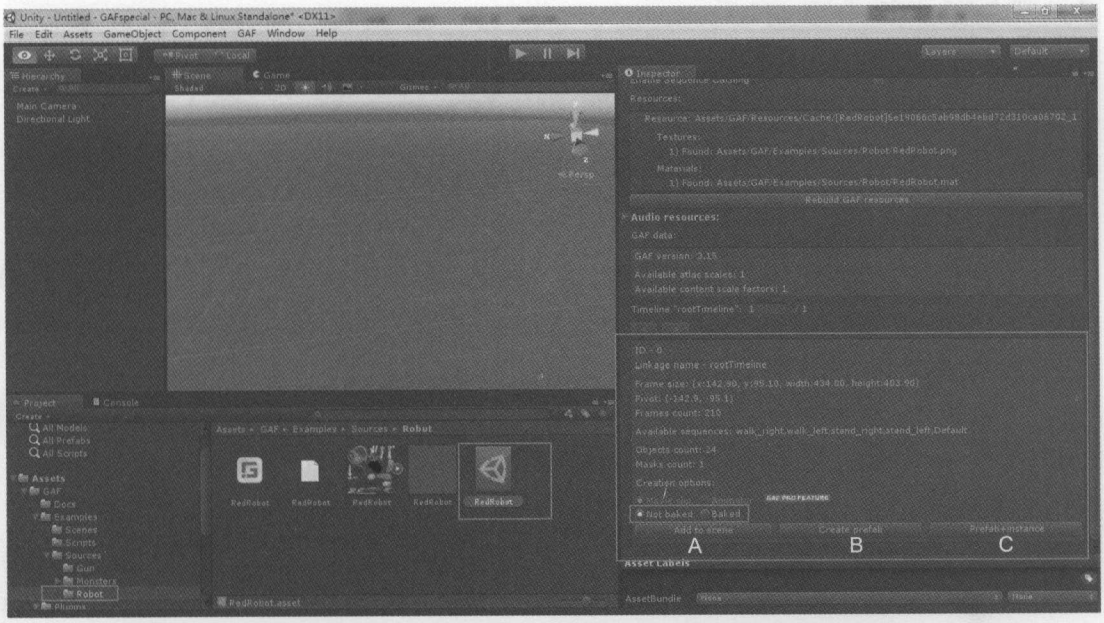


下载好之后将得到 gaf.unitypackage 和 GAF-Converter-5.4.exe 两个文件，exe 文件安装完成之后，能够将 swf 文件直接转化为 Unity 支持的资源，接下来在 Unity 工程中导入 gaf.

unitypackage 插件包，该插件包中包含了能够播放 Flash 资源的代码。

5.2 实现动画播放

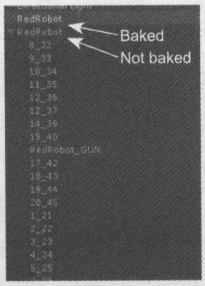
本章采用 GAF 自带的案例来进行讲解，在 Assets 文件夹的 GAF 文件夹下找到 Robot 资源文件。



选择该文件，在 Inspector 面板中找到上图所示位置。

Baked 表示不创建子对象文件，将该 Flash 整合为一个完整的个体。

Not baked 表示创建该 Flash 给部分子对象，形成可拆分对象。



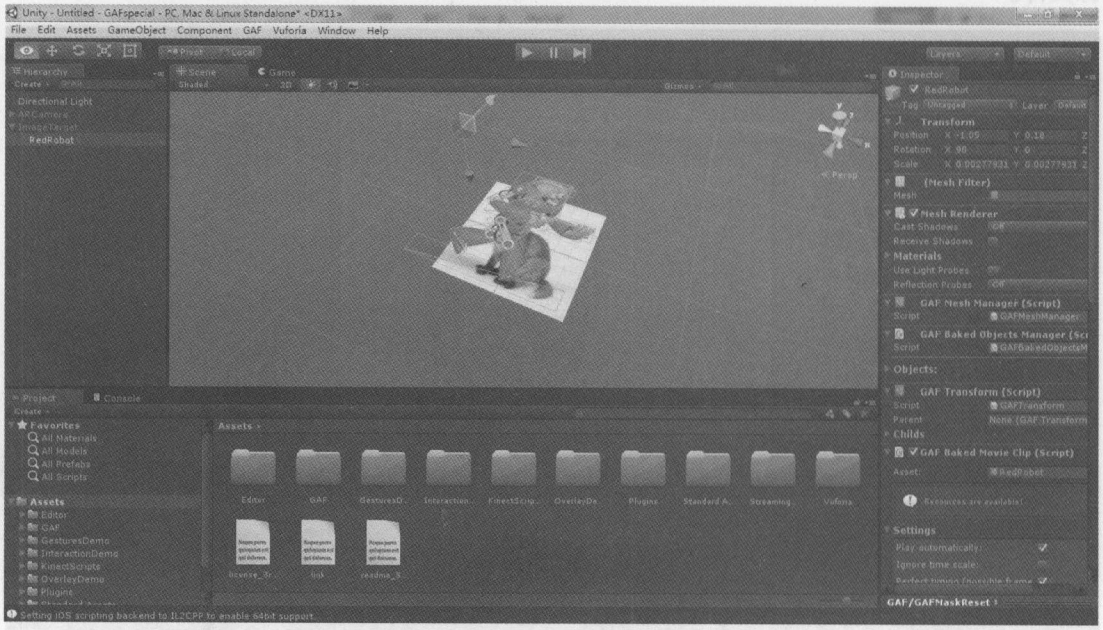
三个按钮的意思分别是：A. 只将该模型生成到场景中；B. 只创建为预制件；C. 在创建为预制件的同时，在场景中也添加一个。

此处我们选择只在场景中创建一个整个的对象，所以选择 Baked 并点击 A 按钮。这样一个 Flash 文件在 Unity 场景中就创建完成了。

5.3 用 AR 方式实现扫描

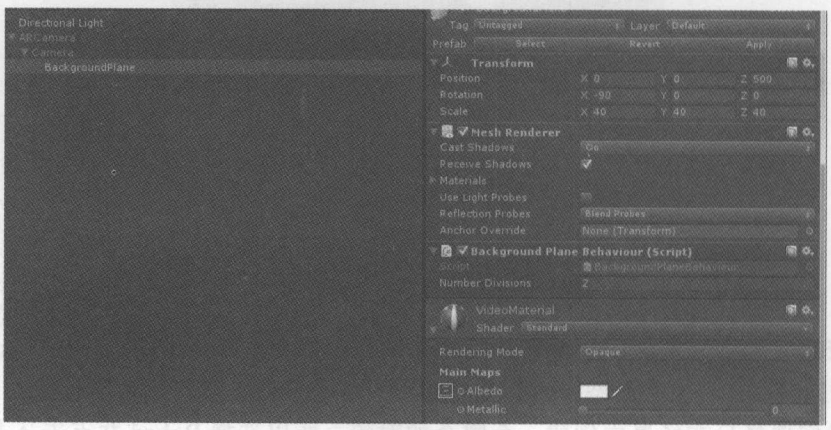
这里我们使用 Vuforia 提供的 SDK 以及数据包，分别导入相应的 SDK 以及在官网加载好的数据包。

将生成的 RedRobot 对象拖到 ImageTarget 下，成为其子对象，然后配置 AR 环境，将 RedRobot 调整到合适的属性位置。



在最新的 Vuforia 版本中，可能会出现无法显示的错误，这时候需要将 ARCamera/Camera/BackgroundPlane 物体的材质球的 Shader 改为 standard。

接下来点击运行进行效果测试，至此，在 Unity 中导入 Flash 动画的教程就结束了。



第6章 AR游戏开发：卡牌对战

本章将讲解AR卡牌对战游戏的详细制作过程，共分为两节，6.1节讲解Vuforia SDK的多图识别生成模型的功能，在此基础上，6.2节讲解卡牌对战功能的实现。

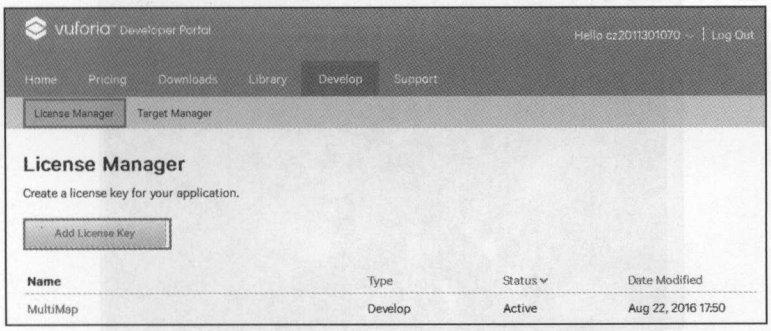
6.1 多图识别生成模型

本书第2章的前两节讲述了利用Vuforia SDK开发单图识别的AR应用案例，本节将基于前面的这部分内容讲述利用Vuforia SDK开发多图识别的AR应用。开始本节内容前需要准备以下内容：

- 两张识别图
- Vuforia SDK

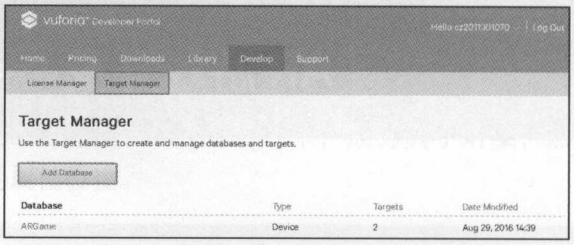
准备好以上内容后，下面开始制作。

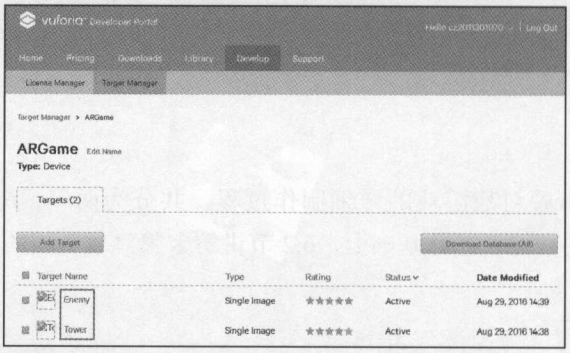
（1）在Vuforia官网的License Manager下创建一个License Key。本节创建了一个名为MultiMap的License Key。



（2）在Target Manager下创建一个Database，将准备好的两张识别图上传到这个

Database 下，然后下载这两个 Targets。本节创建的 Database 是 ARGame，其中包含两个 Targets，分别是 Enemy 和 Tower。





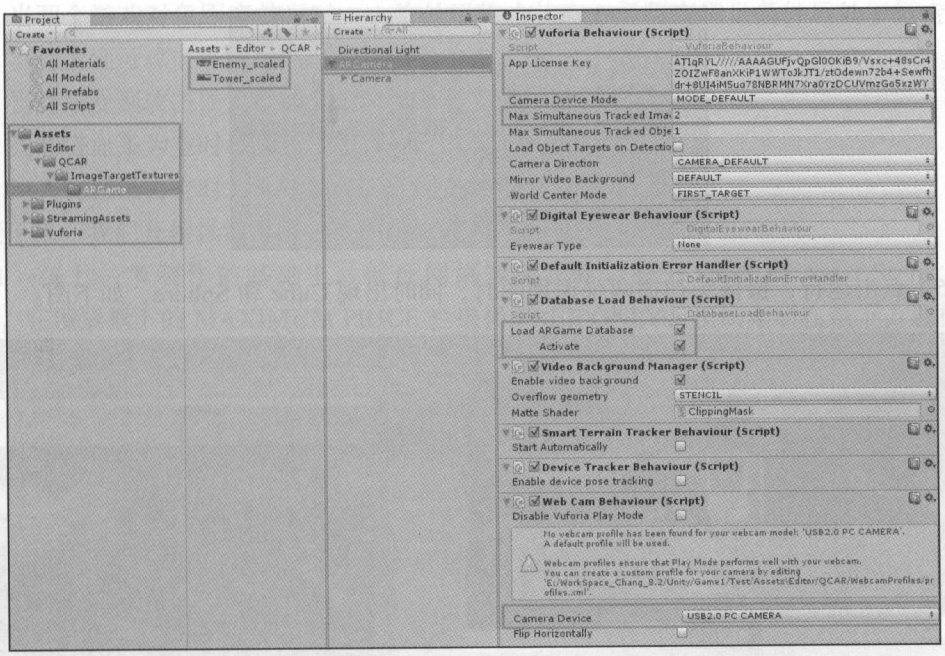
Tower 对应的识别图是：



Enemy 对应的识别图是：



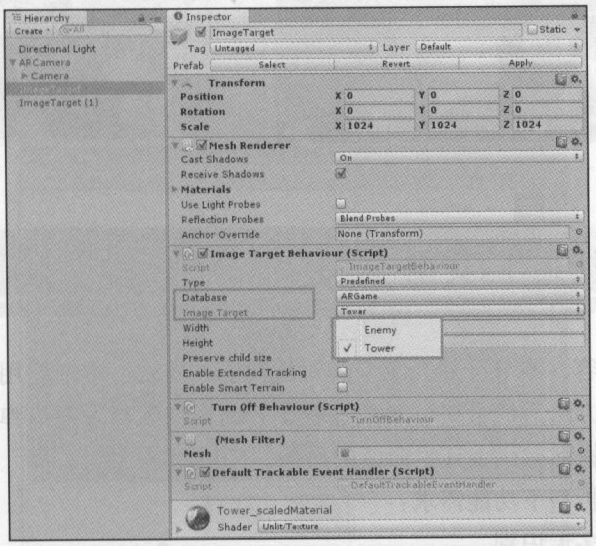
（3）在 Unity 工程中准备好 Vuforia 开发环境，导入下载好的两个 Database，并按下图设置。



注意

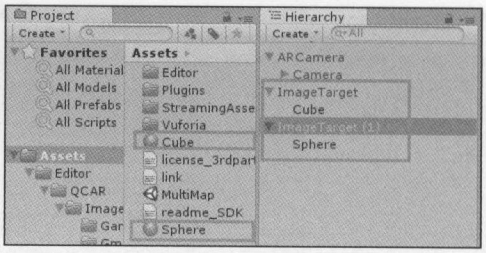
须将 Max Simultaneous Tracked Images 设置成大于 1 的数字（本节设置为 2），这是单图识别和多图识别设置的主要区别。

（4）将 Vuforia 预制件 ImageTarget 拖入 Hierarchy 面板两次，分别设置其 Database 为 Game11 和 Game12。



（5）为测试识别效果，为 ImageTagert 添加一个立方体作为子物体，为 ImageTagert（1）

添加一个球体作为子物体，适当调整两个子物体的大小和颜色。



（6）点击运行，摄像头扫描两张识别图时，同时出现 Cube 和 Sphere，如下图。



6.2 卡牌对战功能的实现

基于上一节的内容，本节开始介绍卡牌对战功能的实现。本节要用到一个高塔模型和一个人物模型，其获取方法后面会讲。对战过程中，高塔向人物发射炮弹，人物可用手中兵器攻击高塔，高塔和人物都有代表其生命值的血条。

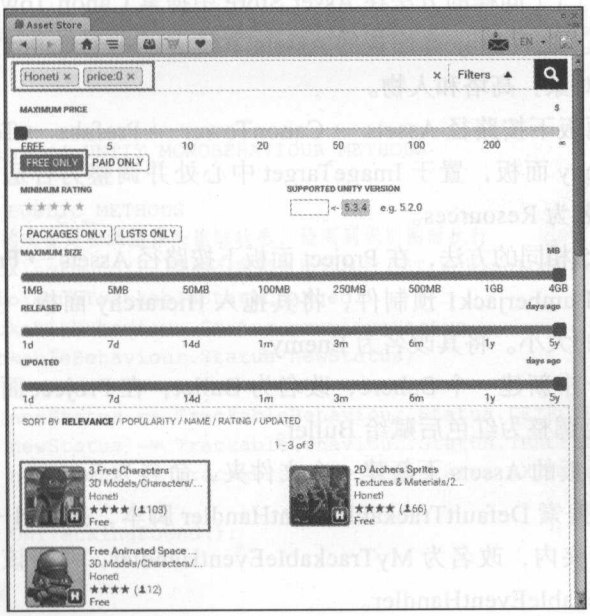
首先，我们来分析一下功能需求：

- 可以进行 AR 多卡识别。
- 当两张卡的距离满足一定条件时，触发对战事件。

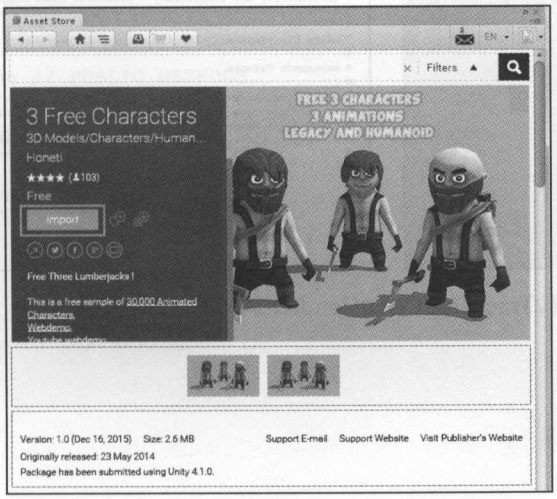
- 炮弹要能追踪到人物，也要能在攻击完成后实施销毁以释放内存资源。
- 当两个游戏对象距离较远时，人物处于静止状态，但距离满足条件后处于行走状态，当行走到可攻击塔的范围内时切换为攻击状态。
- 当高塔受到攻击或人物被袭击时血条会不断消减。
- 当高塔血条为空时，颜色变黑，同时人物状态切换为静止。
- 当人物血条为空时，人物消失，同时高塔停止发射子弹。

接下来，我们开始详细讲解游戏的制作过程。

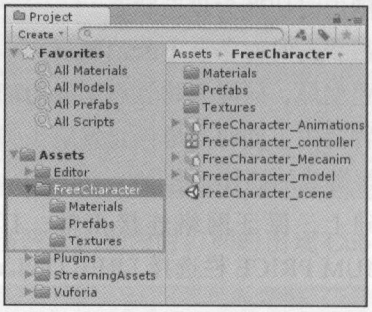
（1）在上一节制作的基础上，保证网络连接正常，打开 Unity 的 Asset Store，搜索 Honeti，在搜索框下的 MAXIMUM PRICE 栏选择 FREE ONLY，按回车后在最下方选择第一个模型，点击下载。



（2）下载完成后选择 import 导入当前工程。



（3）导入后的 Asset 面板下多了一个 Free Character 文件夹，其中包含三个人物模型和相应的动画，本节将使用其中一个人物模型进行讲解。



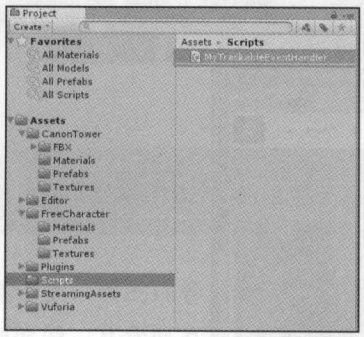
（4）按照与步骤（1）同样的方法在 Asset Store 中搜索 Canon Tower，选择第一个对象，下载并导入当前工程。之后 Project 面板的 Assets 下便多了一个 CanonTower 文件夹。至此，我们获得了两个游戏对象：高塔和人物。

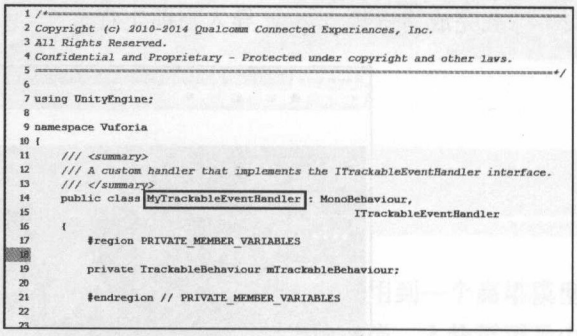
（5）在 Project 面板下按路径 Assets → CanonTower → Prefabs → Tower 找到 Tower 预制件，将其拖入 Hierarchy 面板，置于 ImageTarget 中心处并调整为合适的大小。在 Assets 面板中新建文件夹，命名为 Resources。

（6）按照与步骤 5 相同的方法，在 Project 面板下按路径 Assets → FreeCharactor → Prefabs → Lumberjack1 找到 Lumberjack1 预制件，将其拖入 Hierarchy 面板，置于 ImageTarget（1）中心处并调整为合适的大小。将其改名为 Enemy。

（7）在 Hierarchy 中新建一个 Sphere，改名为 Bullet，在 Project 面板的 Assets 下新建一个 Material，将其颜色调整为红色后赋给 Bullet。

（8）在 Project 面板的 Assets 下新建一个文件夹，命名为 Scripts，用以存放自己写的脚本。在 Assets 面板下搜索 DefaultTrackableEventHandler 脚本，按 Ctrl+D 复制一份，将复制件拖动到 Scripts 文件夹内，改名为 MyTrackableEventHandler，双击该脚本打开编译器，同样将类名改为 MyTrackableEventHandler。





（9）下面来讲解 MyTrackableEventHandler 脚本的框架。

```
using UnityEngine;

namespace Vuforia
{
```

```
public class MyTrackableEventHandler: MonoBehaviour,
ITrackableEventHandler
{
#region PRIVATE_MEMBER_VARIABLES

private TrackableBehaviour mTrackableBehaviour;

#endregion // PRIVATE_MEMBER_VARIABLES

#region UNTIY_MONOBEHAVIOUR_METHODS

void Start()
{
mTrackableBehaviour = GetComponent<TrackableBehaviour>();
if (mTrackableBehaviour)
{
mTrackableBehaviour.RegisterTrackableEventHandler(this);
    }
}

#endregion // UNTIY_MONOBEHAVIOUR_METHODS

#region PUBLIC_METHODS
//该方法的功能是检测摄像头追踪状态，检测到识别图时执行
//OnTrackingFound()，检测不到识别图时执行 OnTrackingLost()
public void OnTrackableStateChanged(
    TrackableBehaviour.Status previousStatus,
    TrackableBehaviour.Status newStatus)
{
    if (newStatus == TrackableBehaviour.Status.DETECTED ||
        newStatus == TrackableBehaviour.Status.TRACKED ||
        newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
    {
        OnTrackingFound();
    }
    else
    {
        OnTrackingLost();
    }
}

#endregion // PUBLIC_METHODS

#region PRIVATE_METHODS

private void OnTrackingFound()
{

}

private void OnTrackingLost()
{

}
```

```
        #endregion // PRIVATE_METHODS
    }
}
```

（10）现在用该脚本来实现成功扫描识别图后动态加载游戏对象 Tower1（之前做好的预制体）。将该脚本挂载到 ImageTarget 上，在 Inspector 面板中将制作好的预制件拖动到 TowerPrefab 属性上，移除脚本 DefaultTrackableEventHandler。

```
using UnityEngine;

namespace Vuforia
{
    public class MyTrackableEventHandler: MonoBehaviour, ITrackableEventHandler
    {
        public GameObject TowerPrefab;
        #region PRIVATE_MEMBER_VARIABLES

        private TrackableBehaviour mTrackableBehaviour;

        #endregion // PRIVATE_MEMBER_VARIABLES


        #region UNTIY_MONOBEHAVIOUR_METHODS

        void Start()
        {
            mTrackableBehaviour = GetComponent<TrackableBehaviour>();
            if (mTrackableBehaviour)
            {
                mTrackableBehaviour.RegisterTrackableEventHandler(this);
            }
        }

        #endregion // UNTIY_MONOBEHAVIOUR_METHODS


        #region PUBLIC_METHODS

        public void OnTrackableStateChanged(
            TrackableBehaviour.Status previousStatus,
            TrackableBehaviour.Status newStatus)
        {
            if (newStatus == TrackableBehaviour.Status.DETECTED ||
                newStatus == TrackableBehaviour.Status.TRACKED ||
                newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
            {
                OnTrackingFound();
            }
            else
            {
                OnTrackingLost();
            }
        }
```

```
        #endregion // PUBLIC_METHODS

        #region PRIVATE_METHODS

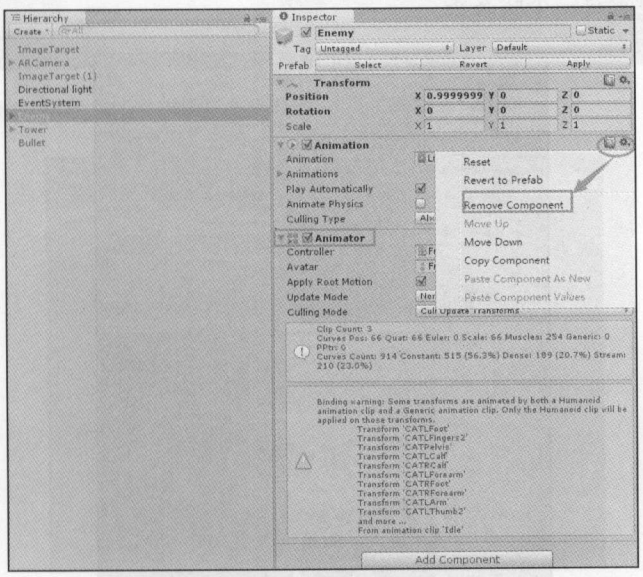

        private void OnTrackingFound()
        {
            GameObject Tower = GameObject.Instantiate (TowerPrefab ) as GameObject;
            Tower.transform .parent = this.transform;
            Tower.transform.position = this.transform.position;
            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " found");
        }


        private void OnTrackingLost()
        {
           TowerController tower = GetComponentInChildren<TowerController>();
           if (tower != null) {
              DestroyImmediate(tower.gameObject);
            }
            Debug.Log("Trackable " + mTrackableBehaviour.TrackableName + " lost");
        }

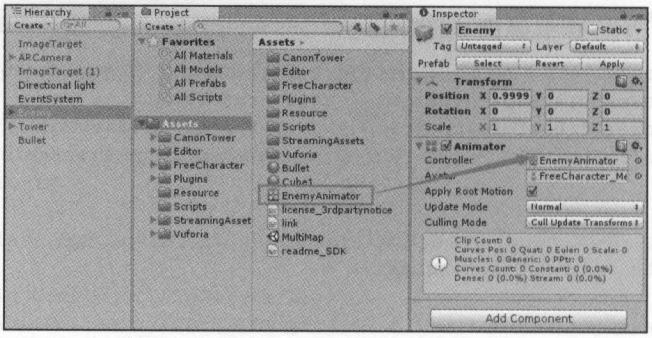
        #endregion // PRIVATE_METHODS
    }
}
```

（11）在 Assets → Scripts 文件夹下新建三个脚本，分别取名为 EnemyController、Tower Controller 和 Bullet，将这三个脚本分别挂载在 Hierarchy 面板中的 Enemy、Tower 和 Bullet 三个游戏对象上，接下来就要分别编辑这三个游戏对象及其脚本以实现对三个游戏对象的控制。

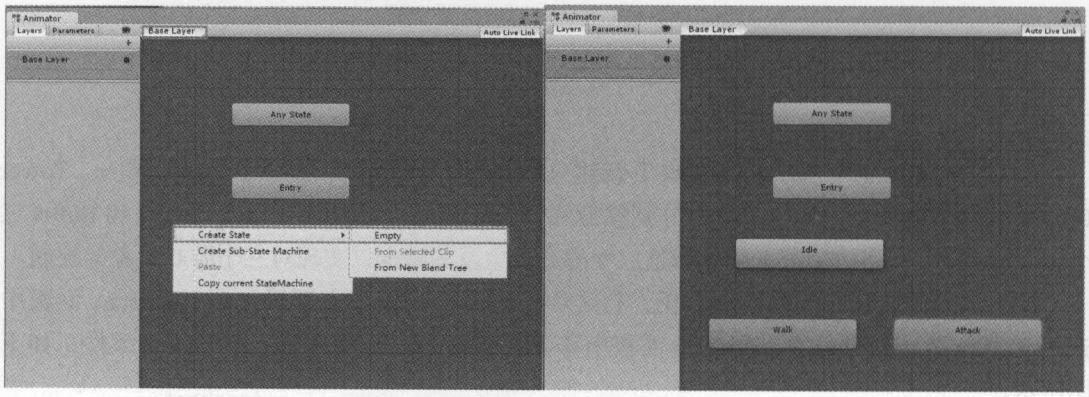
（12）首先来实现 Enemy 的静止、行走和攻击三个动作状态逻辑。在 Hierarchy 面板中选中 Enemy 游戏对象，在 Inspector 面板中移除 Animation 组件，勾选 Animator 组件，如下图所示。



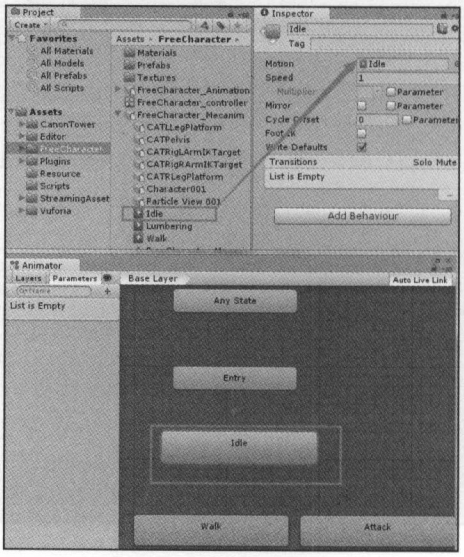
（13）在 Project 面板的 Assets 下单击右键，新建一个 AnimatorController，改名为 EnemyAnimator，在 Hierarchy 面板选中 Enemy，将新建的 EnemyAnimator 拖动到 Inspector 面板 Animator 下的 Controller 中，如下图。



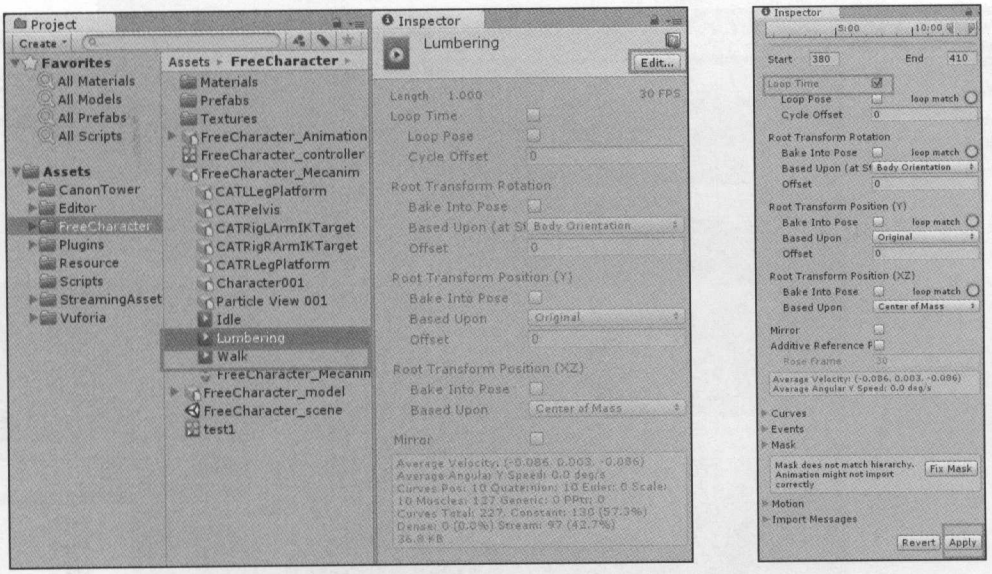
（14）双击 Assets 中的 EnemyAnimator，打开 Animator 窗口，在 BaseLayer 中点击右键，新建三个状态，分别取名为 Idle、Walk、Attack，如下图。



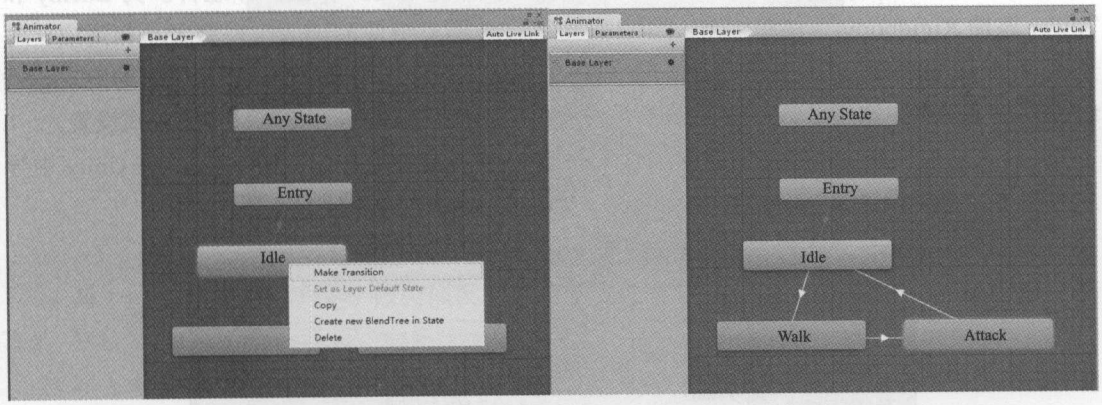
（15）按下图设置每一个状态条。



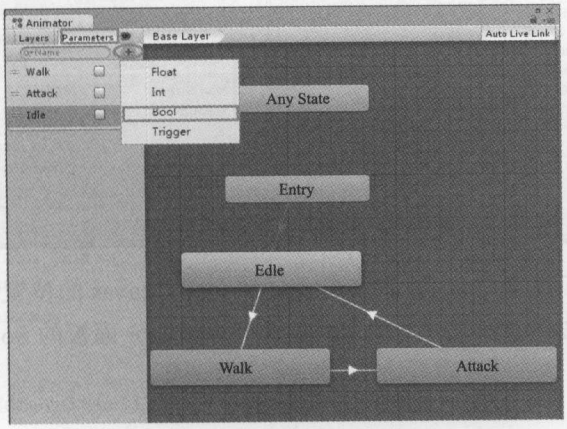
（16）将 Walk 和 Lumbering 两个动画设置为循环播放，具体设置如下图。



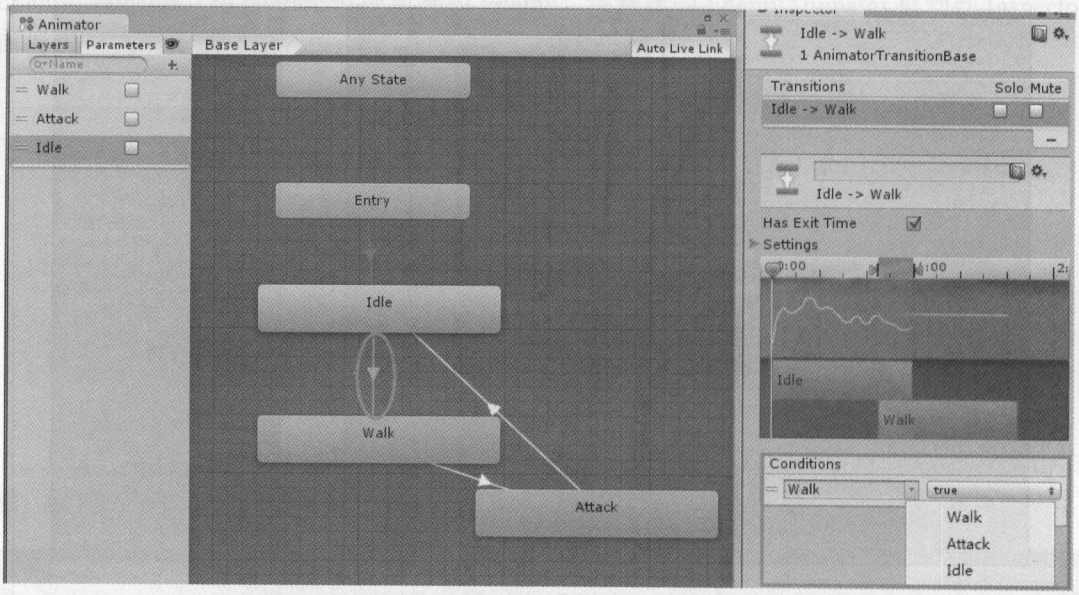
（17）在 Animator 面板的状态条上右键创建如下状态转换。



（18）创建动画转换条件。

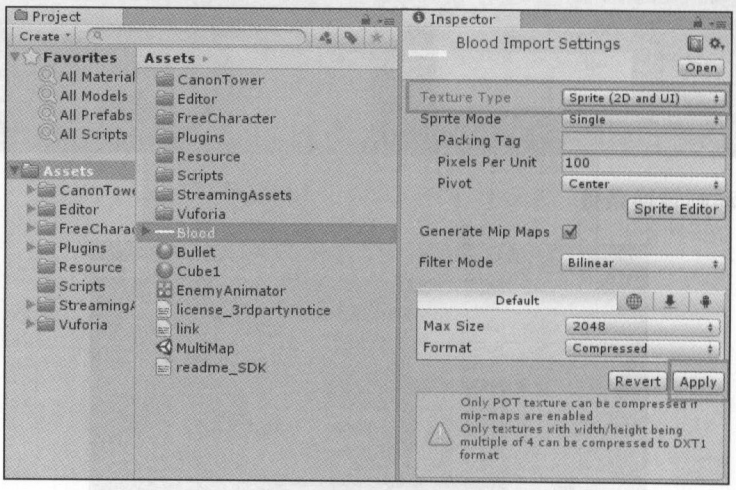


（19）为每个动画转换添加创建好的条件。

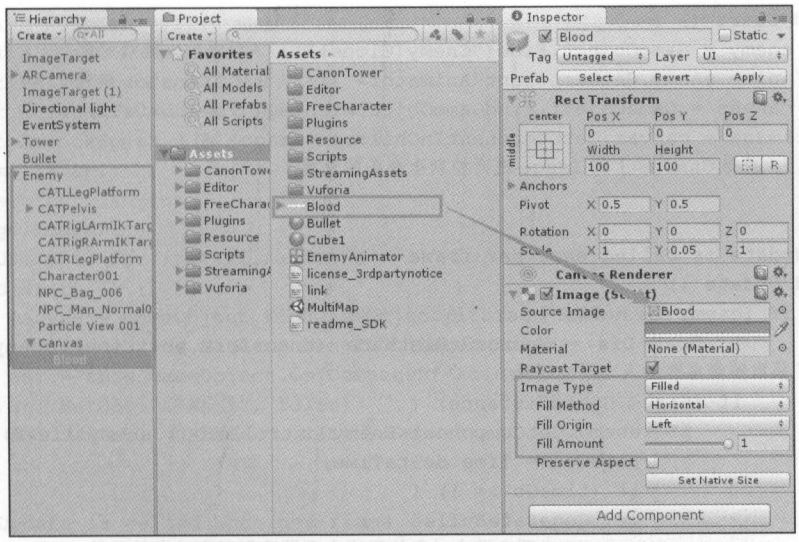


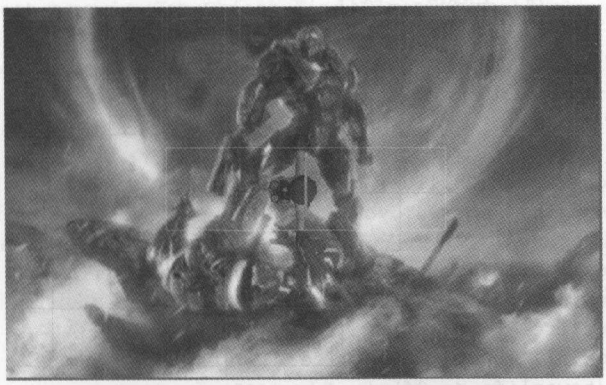
（20）为了进行碰撞检测，需要为游戏对象添加刚体组件和碰撞体组件。为 Enemy 和 Tower 添加 Rigidbody 组件和 CapsuleCollider 组件并调整组件大小以与模型大小相适应，勾选两个 Rigidbody 组件上的 UseGravity 选项，勾选两个 CapsuleCollider 组件上的 IsTrigger 组件，在 Scene 中选中敌人手中的斧子并为其添加以上两个组件，按照同样的方式设置。

（21）创建血条。在 Photoshop 中创建一个白色画布，另存为 Blood.jpg。在 Unity 中导入该图片，将其设置为 Sprite。



（22）在 Enemy 下创建 Image，调整 Image 和对应 Canvas 的位置与大小，为 Canvas 设置一个 Tag 并取名为 EnemyLife，将 Blood 拖动到 Inspector 面板的 SourceImage 上并按下图设置。





（23）仿照上一步为 Tower 做类似设置。为了方便查找，给 Tower 下的 Canvas 添加一个 Tag，取名为 TowerLife。

（24）编写 TowerController 脚本。

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class TowerController : MonoBehaviour {
    public GameObject Bullet;
    EnemyController enemycontroller;
    float timeGo = 0f;
    public float GuardDistance = 300f;
    // 防御塔守卫距离，当距离小于这个值时发射子弹
    public float towerlife = 100f;// 防御塔生命值
    public Transform pos;
    GameObject enemy;
    GameObject canvas;
    Image image;
    Animator ani;
    // Use this for initialization
```

```
    void Start () {
        enemy = GameObject.Find("Enemy(Clone)");// 获得敌人游戏对象
        ani = enemy.GetComponent<Animator> ();// 获得 Animator 组件
        canvas = GameObject.FindGameObjectWithTag("TowerLife");
        image = canvas.GetComponentInChildren<Image> ();
        // 获得 Canvas 上的 Image 进而控制生命条的衰减
    }

    // Update is called once per frame
    void Update () {
        if (enemy != null) {
            float Dis = Vector3.Distance (transform.position, enemy.transform.
position);// 获取防御塔和敌人之间的距离
            if (Dis < GuardDistance) {
                if (enemy.GetComponent<EnemyController>().enemylife>0) {
                    timeGo += Time.deltaTime;
                    if (timeGo >= 1) {
                        GenerateBullet ();
                        // 当两个游戏对象的距离小于塔防御距离时发射子弹
                        timeGo = 0;
                    }
                }
            }
        }
    }
    void  OnTriggerEnter(Collider other)
    {
        // 塔被攻击
        if (other.gameObject.name == "NPC_Tools_Axe_004") {
            Debug.Log(" 防御塔被攻击! ");
            towerlife -= 10f;
            image.fillAmount = towerlife / 100f;
            if (towerlife <= 0f) {
                Renderer[] renderers = GetComponentsInChildren<Renderer> ();
                foreach (Renderer ren in renderers){
                    ren.material.color = Color.black;
                    // 当防御塔的生命值小于等于 0 时将塔的颜色变黑
                }
                ani.SetBool("Idle",true);// 同时使得人物动画切换为静止状态
            }
        }
    }
    void GenerateBullet(){
        GameObject bullet = GameObject.Instantiate(Bullet)as GameObject;
        bullet.transform.position = pos.position;
    }
}
```

（25）编写 EnemyController 脚本。

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
public class EnemyController : MonoBehaviour {
```

```
    public float enemylife = 100f;
    public  float Axedamage = 30f;
    float Walkdistance = 1000f;
    float Attackdistance = 150f;
    float AttackSpeed = 1f;
    Animator ani;
    GameObject canvas;
    Image image;
    // Use this for initialization
    void Start () {
        canvas = GameObject.FindGameObjectWithTag("EnemyLife");
        image = canvas.GetComponentInChildren<Image> ();
        ani = this.gameObject.GetComponent<Animator>();
        ani.SetBool ("Walk", false);
        ani.SetBool ("Attack", false);
    }

    // Update is called once per frame
    void Update () {
        GameObject tower = GameObject.Find("Tower(Clone)");
        //找到游戏对象防御塔
        if (tower != null) {
           float distance = Vector3.Distance (this.transform.position, tower.
transform.position);
            if (distance <= Walkdistance) {
                ani.SetBool ("Walk", true);
             transform.position = Vector3.MoveTowards (this.gameObject.transform.
position, tower.transform.position, AttackSpeed);// 让敌人以 AttackSpeed 值走向防御塔
                if (distance <= Attackdistance) {
                //敌人和防御塔之间的距离小于攻击距离时，停止行走，动画切换为攻击
                    AttackSpeed = 0f;
                    ani.SetBool ("Walk", false);
                    ani.SetBool ("Attack", true);

                }

            }
        }
    }
    void  OnTriggerEnter(Collider other)
    {
        // 当子弹打到敌人身上时的控制
        if (other.gameObject.name == "Bullet(Clone)") {
            enemylife -= Bullet.damage;
            image.fillAmount = enemylife/100f;
            if (enemylife <= 0f) {
          gameObject.SetActive(false);
        // 当敌人的血条值小于等于 0 时，将该游戏对象关闭
            }
        }
    }
}
}
```

（26）编写 Bullet 脚本：

```
using UnityEngine;
using System.Collections;

public class Bullet : MonoBehaviour {
    EnemyController enemycontroller;
    public static float damage = 20;
    GameObject Enemy;
    void Start () {
        Enemy = GameObject.Find("Enemy(Clone)");
    }
    void Update () {
         if(Enemy.activeSelf){this.transform.position = Vector3.MoveTowards(this.
transform.position, Enemy.transform.position,5f);// 子弹以 5 个单位每帧的速度射向敌人
        }
        else
        {
            Destroy(this.gameObject);
        }
        if(transform.position == Enemy.transform.position)
        // 子弹攻击完后自动销毁
            {
                Destroy(this.gameObject);
            }
            }
}
```

（27）将 Hierarchy 中的 Bullet、Enemy 和 Tower 拖回到 Assets 的 Resources 文件夹中作为预制体，并将三者从 Hierarchy 面板中删除。

（28）游戏效果截图如下：



（29）至此，卡牌对战游戏基本制作完成，本游戏还有很多可以提高的部分，大家可以基于前面学习的内容自己探索。

第7章 涂色类AR开发

本章将给大家讲解如何开发涂色类AR。涂色类的AR项目是目前AR市场上相对成功的项目，它对SDK没有太多的要求，我们这里使用的AR SDK是Vuforia。

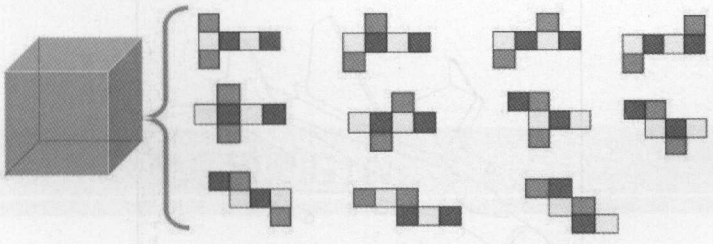
7.1 UV的概念

UV坐标是指所有的图像文件都是一个二维平面。水平方向是U，垂直方向是V，通过二维的UV坐标系可以定位图像上的任意一个像素。

三维物体在显示它的表面纹理时，往往要借助平面贴图来实现，那么一个很重要的问题就是，如何把这个二维平面贴到三维物体的表面呢？

这时候我们就需要对物体进行分UV的处理，“UV”在这里是U，V纹理贴图坐标的简称（它和空间模型的X，Y，Z轴是类似的）。它定义了图片上每个点的位置信息，这些点与3D模型是相互联系的，用以决定表面纹理贴图的位置。UV就是将图像上的每一个点精确对应到模型物体的表面，在点与点之间的间隙位置由软件进行图像光滑插值处理。这就是所谓的UV贴图。

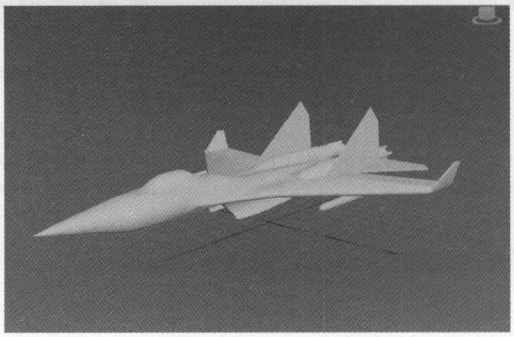
比如正方体模型的展开图：



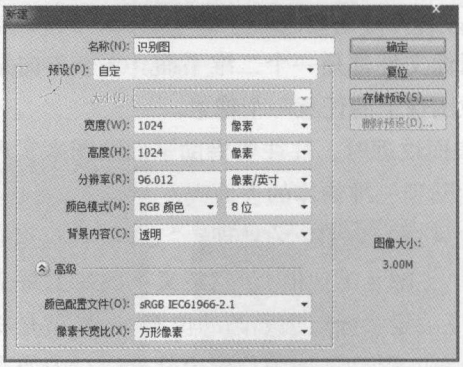
正方体有11种展开图，上面的这些展开图都是正方体的一种UV贴图。

在涂色类的AR项目中，对三维物体的UV处理是非常重要的。我们将在后面给大家具体介绍如何使用3D Max对三维模型的UV进行处理，这里的UV处理和传统的游戏模型UV处理有所不同。

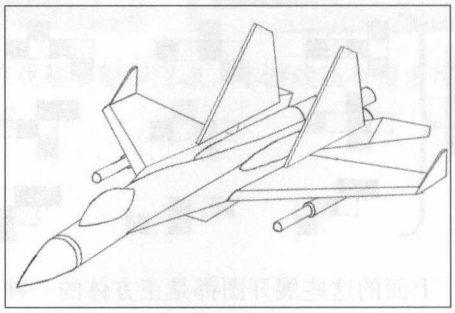
学习下一节内容之前，大家需要准备一个可以在3D Max中打开的三维模型，该模型可自己制作或在网上下载，这里使用的模型是根据苏式战斗机的设计图来简化制作的飞机模型。



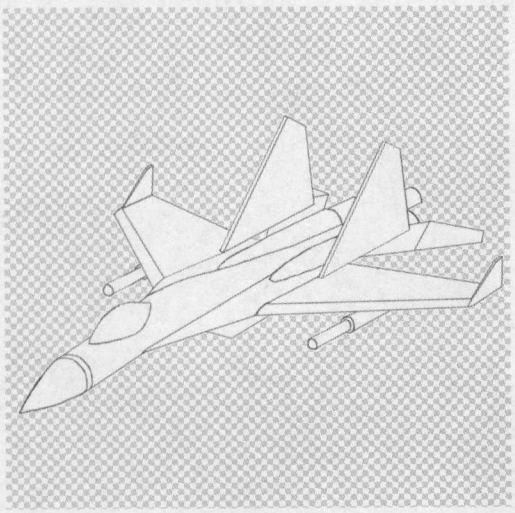
7.2 使用Photoshop绘制涂色的识别图



打开Photoshop，新建一个1024px × 1024px的识别图，这里需要绘制一张用于划分涂色区域的黑白线稿。



这里可根据模型的设定图来绘制涂色用的黑白线稿，这一步使用手绘板可以提高效率。用选区的方式将多余的白色部分删除得到以下图片。



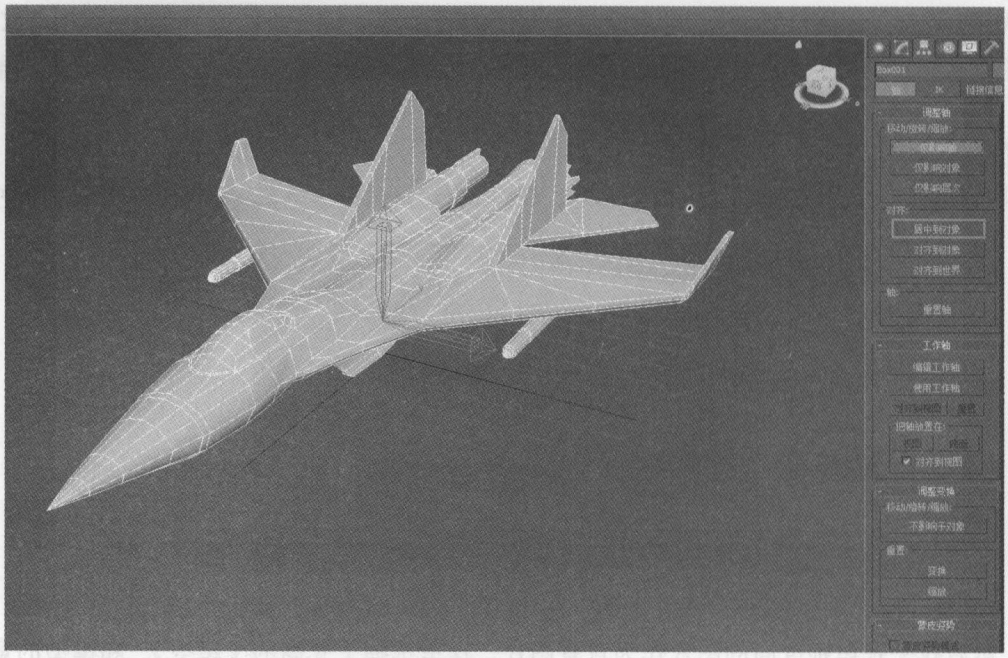
然后找一张识别度高的图片作为背景图片，这里选择了梵高的《星空》，和模型的意义也很匹配。



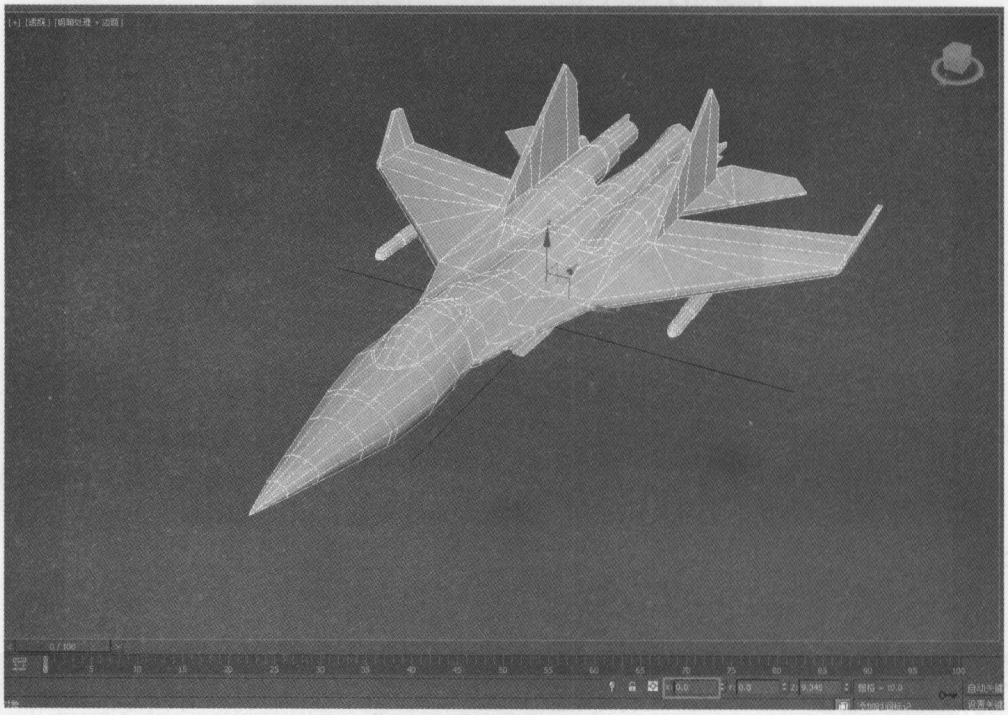
这张图将可以作为我们的AR识别图，接下来要根据这张识别图来对模型的UV进行具体处理。

7.3 使用3D Max处理模型的UV

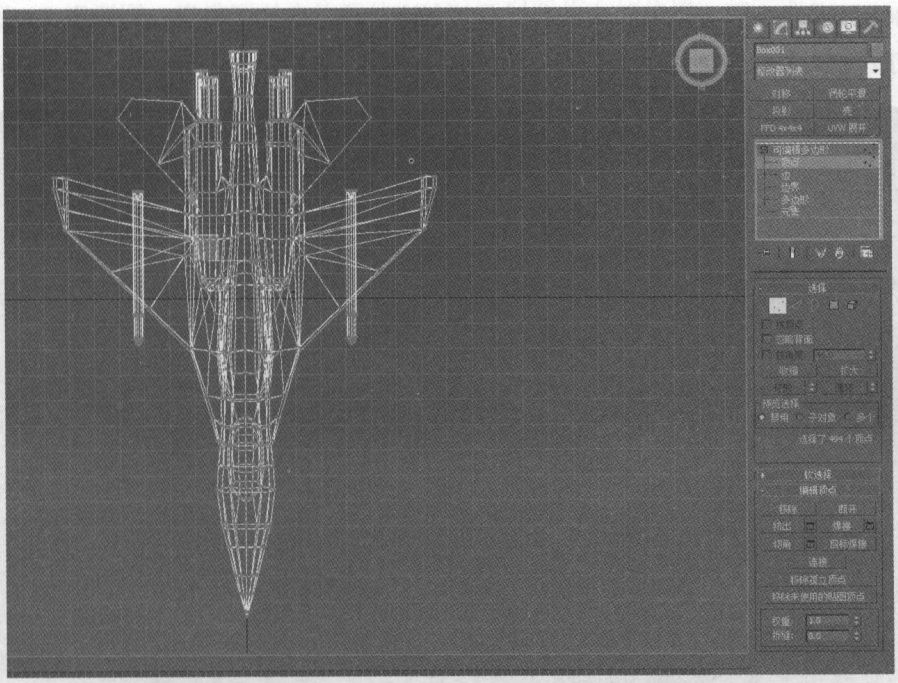
打开3D Max，如果模型是两边对称的，先把模型的轴心居中。



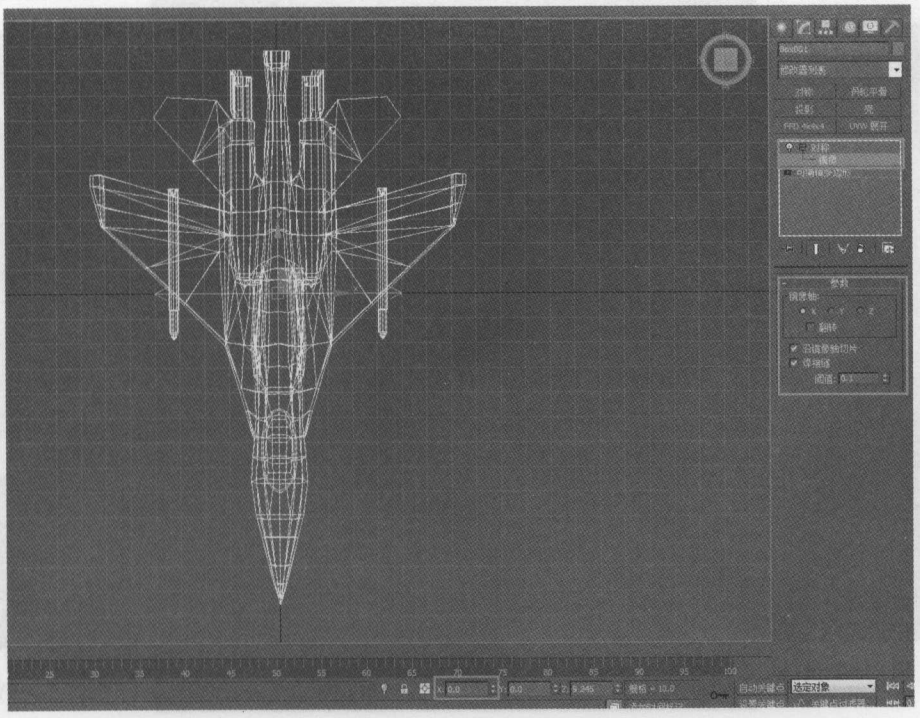
然后退出轴心编辑模式，在移动模式下将模型对称的轴的位置归零，这里飞机的模型对称轴是 X 轴，所以我们将模型沿 X 方向的位置归零。



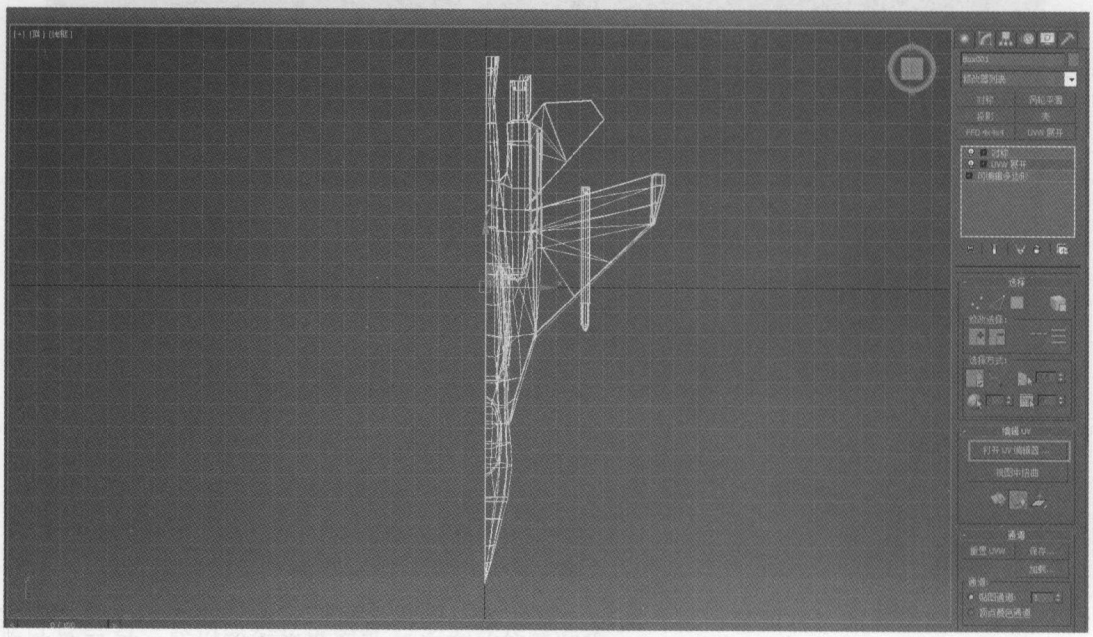
切换到顶视图，在顶点的级别下选中并删除一半的顶点（切记：对称线上的顶点不能删除）。



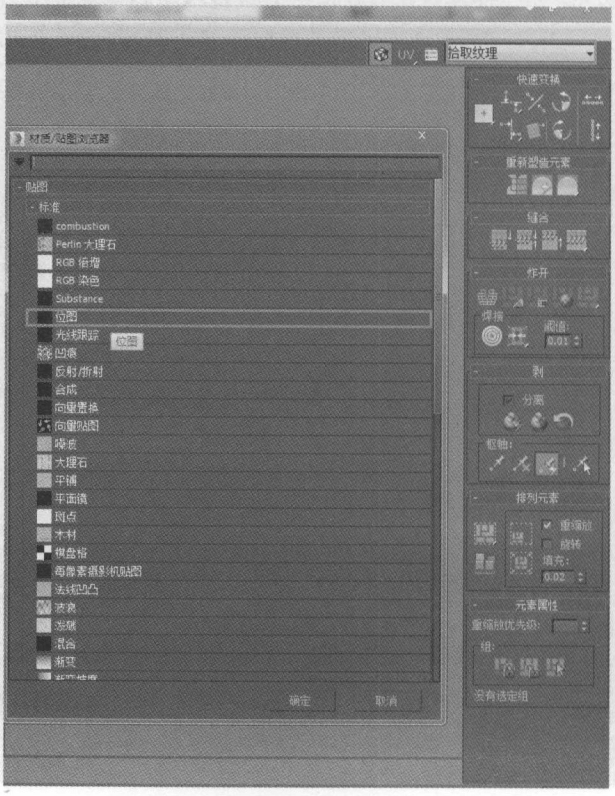
然后给模型添加对称的修改器，路径为修改器→网格编辑→对称，或者在修改器列表的下拉列表框中选择相应选项，也可以按下键盘上的“X”键，在弹出的查找框中输入“对称”，然后选择修改器。添加对称修改器后，调整对称修改器的参数如下图所示，对称轴选择模型的对称轴，如果模型消失，试着勾选翻转的选项框，一定要把镜像级别沿 X 方向的位置归零。



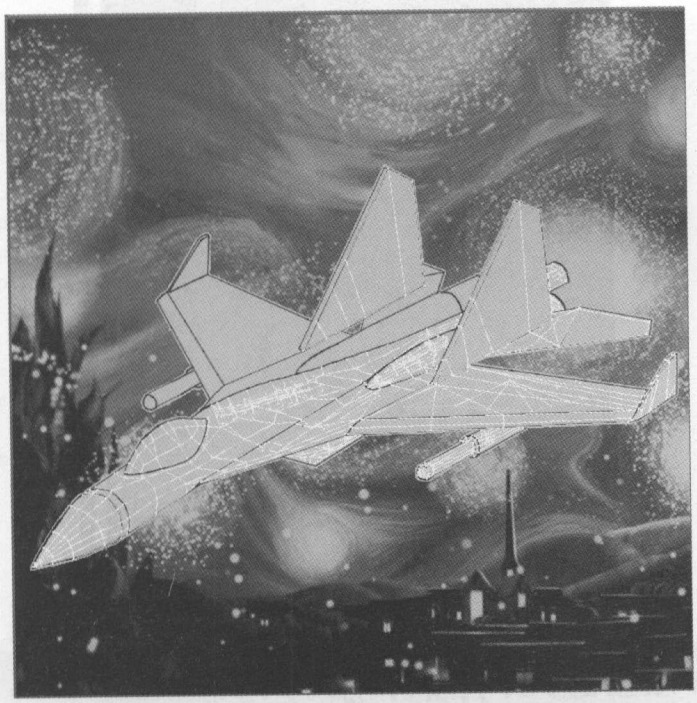
在对称修改器的下面添加“UVW 展开”的修改器，路径为修改器→UV 坐标→UVW 展开，打开 UV 编辑器。



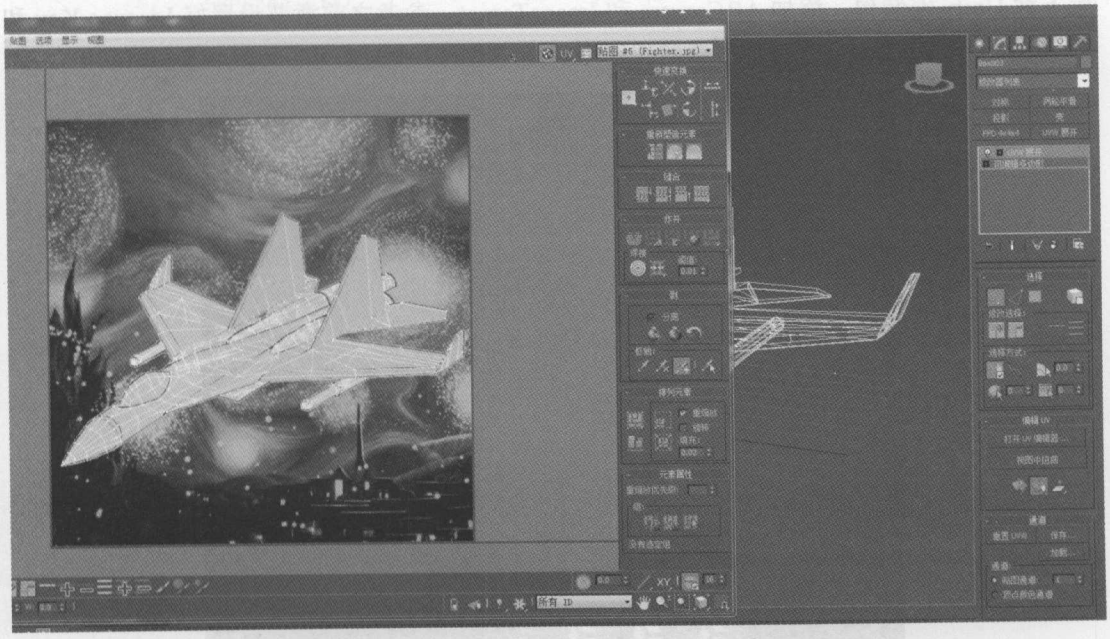
拾取纹理，选择位图，打开上一节制作的 AR 识别图。



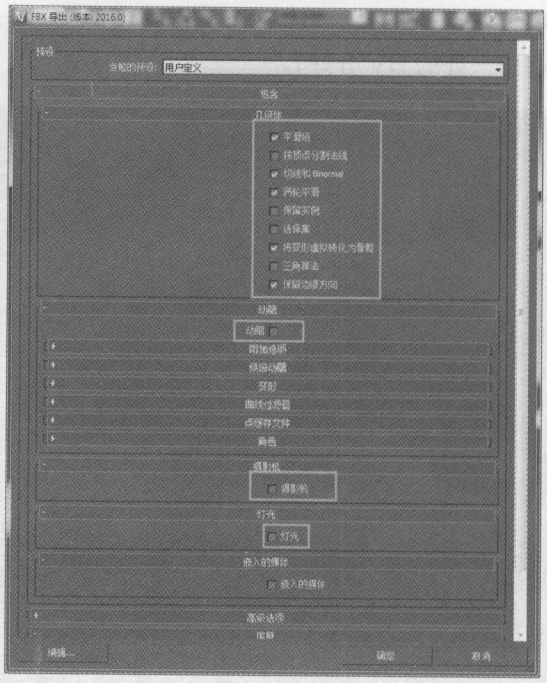
根据AR识别图，将模型对应位置的UV对齐到图片上的位置，如下图所示（这个过程会很费时间，需要耐心将每一处的UV分好）。



做好之后，我们将所有修改器塌陷，再添加UVW展开的修改器，打开UV编辑器，将另一半机翼的UV也对应起来，其他部分不用改变。

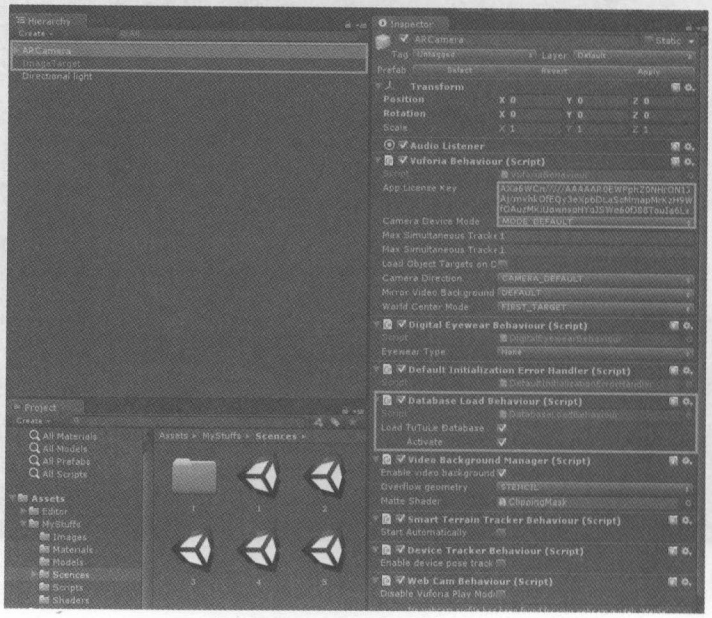


做好之后将模型导出成.fbx格式的文件，导出设置如下图。



7.4 搭建 Vuforia AR 开发环境

进入 Vuforia 官网下载 SDK，创建项目和数据库，上传我们之前制作好的识别图，然后下载 Unity 安装包，添加 ARCamera 和 ImageTarget，参考之前章节设置好 License Key 和 Database，ImageTarget 选择你下载的识别图。最终做到下面这一步。



这里采用Shader实现从屏幕截图中获取我们所需的贴图。涂色类AR的Shader有很多种，下面给大家介绍其中一个Shader。新建一个Shader文件，将原本的内容删除，添加以下的代码：

```
Shader " Customer/TuSeARShader" {
    Properties {
        _MainTex ("Base (RGB)", 2D) = "white" {}

        _Uvpoint1("point1", Vector) = (0 , 0 , 0 , 0)
        _Uvpoint2("point2", Vector) = (0 , 0 , 0 , 0)
        _Uvpoint3("point3", Vector) = (0 , 0 , 0 , 0)
        _Uvpoint4("point4", Vector) = (0 , 0 , 0 , 0)

    }
    SubShader {
        Tags { "Queue"="Transparent" "RenderType"="Transparent" }
        LOD 200
        Pass{
            Blend SrcAlpha OneMinusSrcAlpha
             CGPROGRAM
             #pragma vertex vert
             #pragma fragment frag
            #include "UnityCG.cginc"
            sampler2D _MainTex;
            float4 _MainTex_ST;
            float4 _Uvpoint1;
            float4 _Uvpoint2;
            float4 _Uvpoint3;
            float4 _Uvpoint4;
            float4x4 _VP;

            struct v2f {
                float4  pos : SV_POSITION;
                float2  uv : TEXCOORD0;
                float4  fixedPos : TEXCOORD2;
            } ;


            v2f vert (appdata_base v)
            {
                v2f o;
                o.pos = mul(UNITY_MATRIX_MVP,v.vertex);
                o.uv = TRANSFORM_TEX(v.texcoord,_MainTex);

                float4 top = lerp(_Uvpoint1, _Uvpoint3, o.uv.x);
                float4 bottom = lerp(_Uvpoint2, _Uvpoint4, o.uv.x);
                float4 fixedPos = lerp(bottom, top, o.uv.y);
                o.fixedPos = ComputeScreenPos(mul(UNITY_MATRIX_VP, fixedPos));
                return o;
            }
            float4 frag (v2f i) : COLOR
            {

                float4 top = lerp(_Uvpoint1, _Uvpoint3, i.uv.x);
```

```
                float4 bottom = lerp(_Uvpoint2, _Uvpoint4, i.uv.x);
                float4 fixedPos = lerp(bottom, top, i.uv.y);
                fixedPos = ComputeScreenPos(mul(_VP, fixedPos));
                return tex2D(_MainTex, fixedPos.xy / fixedPos.w);

            }
            ENDCG

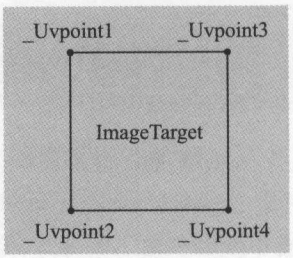
        }
    }

}
```

我们需要向这个 Shader 中的 6 个变量传递正确的值，Shader 会帮我们处理好模型和识别图的对应关系，下面一一介绍这 6 个变量。

```
float4 _Uvpoint1;
float4 _Uvpoint2;
float4 _Uvpoint3;
float4 _Uvpoint4;
```

首先是以上 4 个变量，我们要向它们传递 4 个 Vector4 类型的数据，这 4 个 Vector4 数据是 ImageTarget 在世界空间中的位置，它们的对应关系如下图所示。



由于位置信息是 Vector3，所以要在末尾补 1.0f 凑成 Vector4。

```
float4x4 _VP;
```

我们需要给“_VP”传递的是 GPU 的投影矩阵和世界坐标到相机的矩阵的乘积：

```
sampler2D _MainTex;
```

MainTexture 需要传递的是整个屏幕的截图。

以上 6 个参数正确传递之后，我们的涂色类 AR 项目的核心功能就实现了，下一节将展示具体的代码实现！

7.5 核心功能之通过 C# 脚本获得变量

在 Unity 中创建一个 C# 脚本，命名为 ScreenTexture。以下为该脚本的具体代码（基础较薄弱的同学可以直接跳过，后面有一步步的讲解）：

```
using UnityEngine;
using System.Collections;

public class ScreenTexture : MonoBehaviour
{

    public GameObject myFighter;

    private Texture2D screenTexture;

    private Vector3 coordinateTopLeft_pl_w;
    private Vector3 coordinateTopRight_pl_w;
    private Vector3 coordinateBottomLeft_pl_w;
    private Vector3 coordinateBottomRight_pl_w;


    public GameObject plane;
    private Vector2 planeWH;

    void Start ()
    {

        screenTexture = new Texture2D(Screen.width, Screen.height, TextureFormat.
RGB24, false);
    }

    public void OnScreenShotButtonClick()
    {

        planeWH = new Vector2(plane.GetComponent<MeshFilter>().mesh.bounds.size.x,
plane.GetComponent<MeshFilter>().mesh.bounds.size.z)  *5*0.5f;// plane.transform.
localScale.x * plane.transform.parent.localScale.x *0.5f;


        coordinateTopLeft_pl_w = plane.transform.parent.position + new Vector3(-plane
WH. x, 0, planeWH.y);

        coordinateBottomLeft_pl_w = plane.transform.parent.position + new Vector3
(-planeWH.x, 0, -planeWH.y);

        coordinateTopRight_pl_w = plane.transform.parent.position + new Vector3
(planeWH.x, 0, planeWH.y);

        coordinateBottomRight_pl_w = plane.transform.parent.position + new Vector3
(planeWH.x, 0, -planeWH.y);



        myFighter.GetComponent<Renderer>().material.SetVector(
              "_Uvpoint1",new Vector4(coordinateTopLeft_pl_w.x,coordinateTopLeft_
pl_w.y,coordinateTopLeft_pl_w.z,1.0f)
        );
        myFighter.GetComponent<Renderer> ().material.SetVector (
```

```
                    "_Uvpoint2", new Vector4 (coordinateBottomLeft_pl_w.x,
coordinateBottomLeft_pl_w.y, coordinateBottomLeft_pl_w.z, 1.0f)
            );
            myFighter.GetComponent<Renderer> ().material.SetVector (
                    "_Uvpoint3", new Vector4 (coordinateTopRight_pl_w.x,
coordinateTopRight_pl_w.y, coordinateTopRight_pl_w.z, 1.0f)
            );
            myFighter.GetComponent<Renderer> ().material.SetVector (
                    "_Uvpoint4", new Vector4 (coordinateBottomRight_pl_w.x,
coordinateBottomRight_pl_w.y, coordinateBottomRight_pl_w.z, 1.0f)
            );
            Matrix4x4 P = GL.GetGPUProjectionMatrix (Camera.main.projectionMatrix, false);
            Matrix4x4 V = Camera.main.worldToCameraMatrix;
            Matrix4x4 VP = P * V;

            myFighter.GetComponent<Renderer> ().material.SetMatrix ("_VP",VP);

            screenTexture.ReadPixels(new Rect(0, 0, screenWidth, screenHeight), 0, 0);


            screenTexture.Apply();
            myFighter.GetComponent<Renderer>().material.mainTexture = textureShot;
        }

    }
```

接下来，我们来看这个脚本的内容。根据上一节的内容，需要给 Shader 传递 6 个参数，那么我们先来分析这 6 个变量。

为了获得 MainTexture，我们需要申请一个 Texture2D 的变量：

```
private Texture2D screenTexture;
```

在 Start() 函数中对它进行初始化：

```
screenTexture = new Texture2D(Screen.width, Screen.height, TextureFormat.RGB24,
false);
```

Screen.width 和 Screen.height 是图片的宽和高，TextureFormat.RGB24 是图片的模式，false 是不使用纹理分级。

在屏幕截取函数里得到整个屏幕的图像：

```
screenTexture.ReadPixels(new Rect(0, 0, screenWidth, screenHeight), 0, 0);
```

```
void ReadPixels (source : Rect, destX : int, destY : int, recalculateMipMaps :
bool = true)
```

读取屏幕像素信息并存储为纹理数据，这将从当前处于激活状态的 RenderTexture 或视图（由 source 指定）复制一个由 destX 和 destY 指定的矩形像素区域。这两个坐标使用像素空间坐标（0，0）表示屏幕左下角。

如果 recalculateMipMaps 设置为真，那么这个贴图的 mipmaps 就会更新。

如果 recalculateMipMaps 设置为假，则需要调用 Apply 重新计算它们。

这个函数只工作在格式为ARGB32和RGB24的纹理上，另外这个纹理的导入设置需要设置为Is Readable（可读）。

获取截图之后一定要调用

```
screenTexture.Apply();
```

应用全部的像素调用，这样就获得了第一个参数，然后获得GPU的投影矩阵和世界坐标到相机的矩阵的乘积。

GPU的投影矩阵为：

```
Matrix4x4 P = GL.GetGPUProjectionMatrix (Camera.main.projectionMatrix, false);
```

世界坐标到相机的矩阵为：

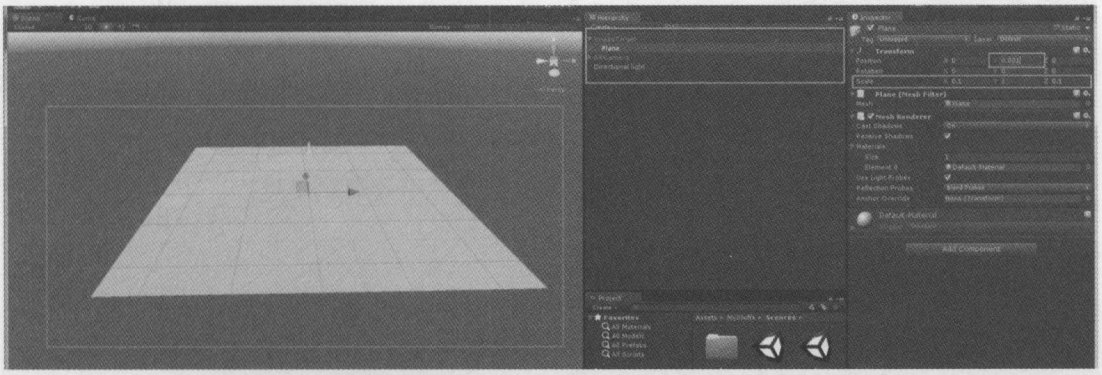
```
Matrix4x4 V = Camera.main.worldToCameraMatrix;
```

将它们相乘：

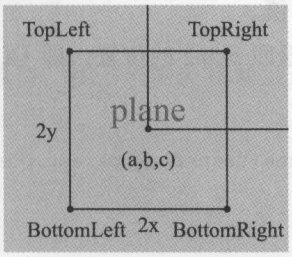
```
Matrix4x4 VP = P * V;
```

然后获得4个Vector4类型的数据。4个类型的数据并不能直接从ImageTarget上获得，所以我们采取的做法是得到它们的近似值。

在Unity中创建一个plane物体，将它放在ImageTarget下面作为其子物体，调整plane物体的X和Z的缩放值来保证它和ImageTarget大小一样，然后将plane的Y轴位置向上移动0.001，这里的数值应该尽可能小，因为我们要用plane的4个点位置来近似地代替ImageTarget的4个点的位置信息。



为了得到这4个点的位置信息，设plane的位置为（a，b，c），宽为2x，高为2y。



由简单的数学运算可得到：

```
TopLeft : (a-x,b,c+y)
TopRight: (a+x,b,c+y)
BottomLeft: (a-x,b,c-y)
BottomRight: (a+x,b,c-y)
```

我们首先声明两个变量：

```
public GameObject plane;
private Vector2 planeWH;
```

前者用于存储场景中的 plane 物体，后者用于存储场景中 plane 物体的 X 轴和 Z 轴的距离的一半。

再申请 4 个 Vector3 来存放 4 个点的位置。

```
private Vector3 coordinateTopLeft_pl_w;
private Vector3 coordinateTopRight_pl_w;
private Vector3 coordinateBottomLeft_pl_w;
private Vector3 coordinateBottomRight_pl_w;
```

在屏幕截取的函数中，先得到 plane 物体的 X 轴和 Z 轴距离的一半，即得到 planeWH 的值。

```
planeWH = new Vector2
(
plane.GetComponent<MeshFilter>().mesh.bounds.size.x*
plane.transform.localScale.x *
plane.transform.parent.localScale.x,
plane.GetComponent<MeshFilter>().mesh.bounds.size.z*
plane.transform.localScale.z * plane.transform.parent.localScale.z
) *0.5f;

plane.GetComponent<MeshFilter>().mesh.bounds.size.x
```

上述语句的返回值是 plane 这个物体的网格的包围体沿 X 轴的长度。

注意

需要给 Shader 传递的是 ImageTarget 世界中的位置，这里用 plane 子物体的位置来做近似，所以必须用 plane 相对世界的缩放值。而 plane 物体的缩放值是相对于 ImageTarget 的，所以要用 plane 的缩放值乘以其父物体的缩放值，得到的才是相对世界的缩放值。

最后再乘 0.5f，取得它的一半值。

Plane.transform.position 可以返回 plane 的位置，即（a,b,c）的值。

套用上面的数学公式可得到：

```
coordinateTopLeft_pl_w = plane.transform.parent.position + new Vector3(-planeWH.x,
0, planeWH.y);
```

```
    coordinateBottomLeft_pl_w = plane.transform.parent.position + new Vector3(-
planeWH.x, 0, -planeWH.y);

    coordinateTopRight_pl_w = plane.transform.parent.position + new Vector3(planeWH.x,
0, planeWH.y);

    coordinateBottomRight_pl_w = plane.transform.parent.position + new
Vector3(planeWH.x, 0, -planeWH.y);
```

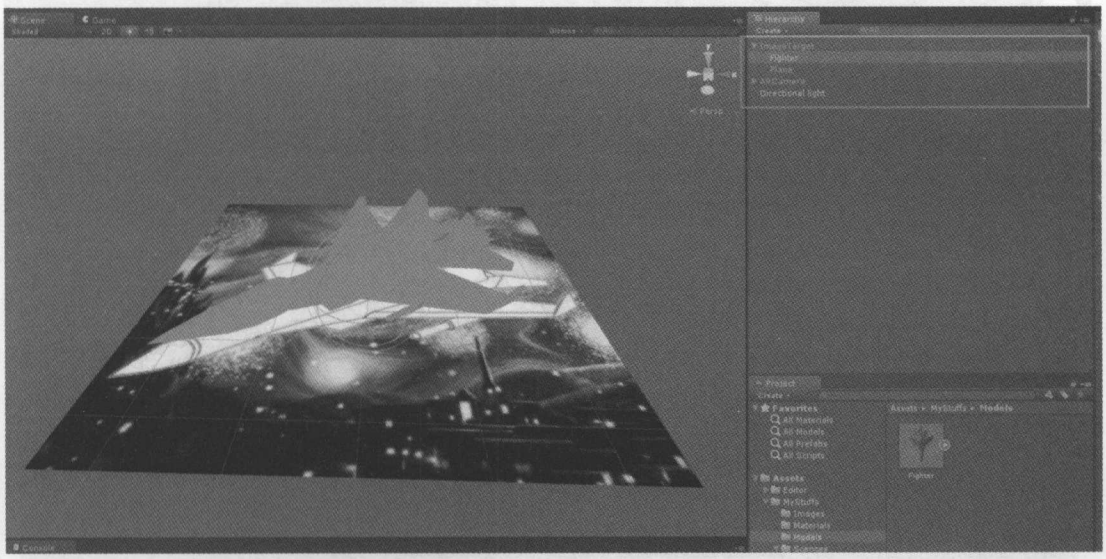
7.6 核心功能之通过C#脚本向Shader传递变量

获取了这6个变量之后，需要向Shader中传递这6个变量的值。

首先申请一个共有的GameObject物体，这个物体就是我们的模型物体。

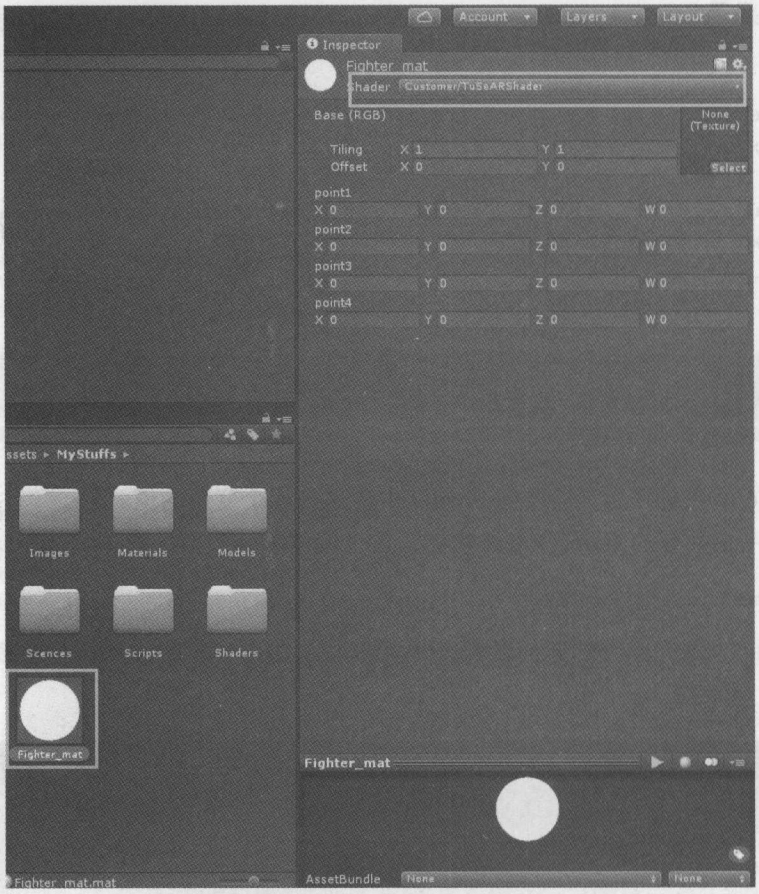
```
public GameObject myFighter;
```

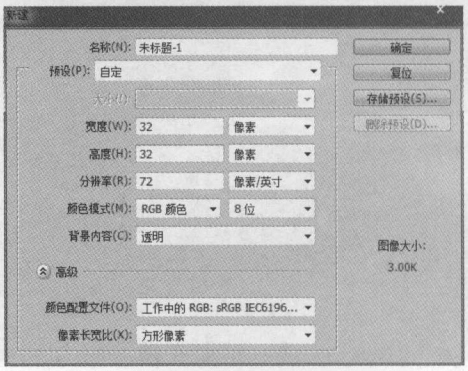
保存之后回到Unity，将导入的模型放在ImageTarget下面，并调整它的大小和旋转方向。



新建一个材质球，改名为Fighter_mat，将它的Shader改为我们之前写的Shader，路径是Customer/TuSeARShader。

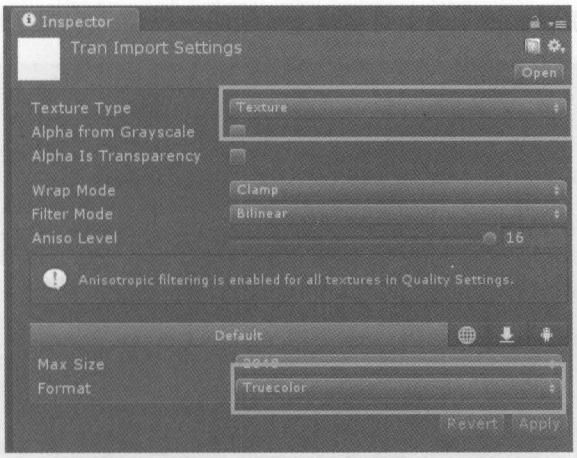
将这个材质球赋值给我们的模型，接下来需要将模型透明显示，在Photoshop中创建一个透明的贴图，大小为32px × 32px即可。



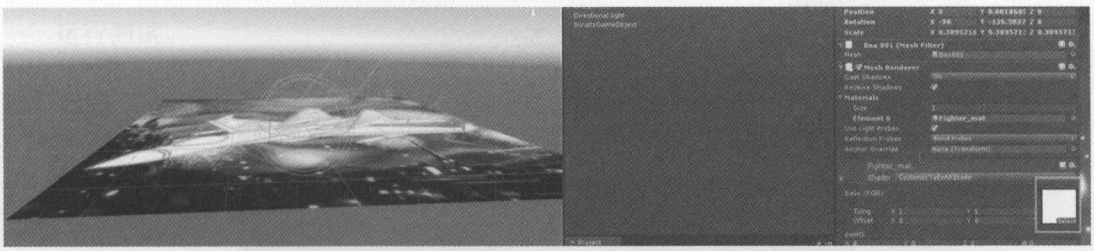




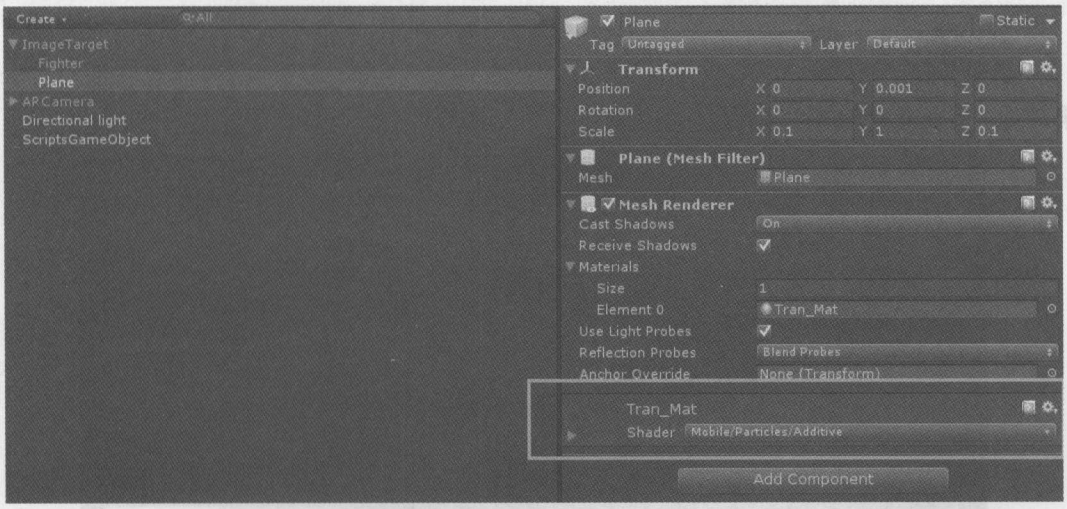
将图层的不透明度改为 0%，然后保存成 .png 格式，导出到 Unity 中。



图片的属性如上图所示，将该图片赋值给模型的 MainTexture，即可实现透明显示。



用同样的方法实现 plane 物体的透明显示，Shader 使用的是 Mobile/Particles/Additive。



新建一个空物体，命名为 ScriptsGameObject，将上面的脚本拖给它，将 plane 和模型物体拖给脚本。



回到脚本中，在屏幕截图函数中对变量进行赋值。

Vector4 变量的赋值：

```
        myFighter.GetComponent<Renderer>().material.SetVector(
                    "_Uvpoint1",new Vector4(coordinateTopLeft_pl_w.x,coordinateTopLeft_
pl_w.y,coordinateTopLeft_pl_w.z,1.0f)
            );//左上
            myFighter.GetComponent<Renderer> ().material.SetVector (
                    "_Uvpoint2", new Vector4 (coordinateBottomLeft_pl_w.x,
coordinateBottomLeft_pl_w.y, coordinateBottomLeft_pl_w.z, 1.0f)
            );//左下
            myFighter.GetComponent<Renderer> ().material.SetVector (
                    "_Uvpoint3", new Vector4 (coordinateTopRight_pl_w.x,
coordinateTopRight_pl_w.y, coordinateTopRight_pl_w.z, 1.0f)
            );//右上
            myFighter.GetComponent<Renderer> ().material.SetVector (
                    "_Uvpoint4", new Vector4 (coordinateBottomRight_pl_w.x,
coordinateBottomRight_pl_w.y, coordinateBottomRight_pl_w.z, 1.0f)
            );//右下
```

矩阵的赋值：

```
myFighter.GetComponent<Renderer> ().material.SetMatrix ("_VP" ,VP);
```

MainTexture 的赋值：

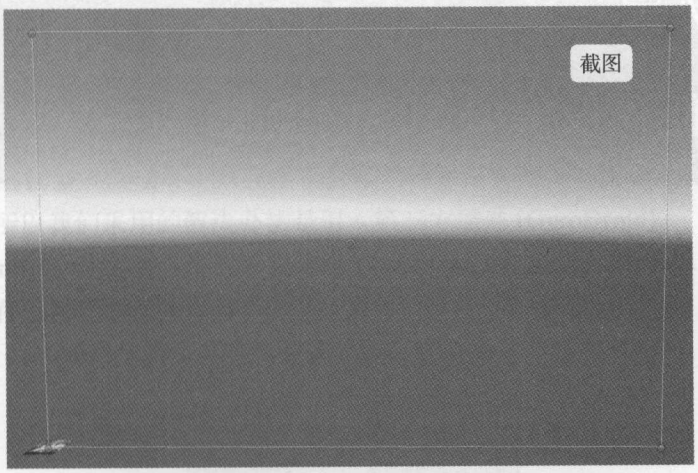
```
myFighter.GetComponent<Renderer>().material.mainTexture = textureShot;
```

7.7 在 Unity 中完善项目

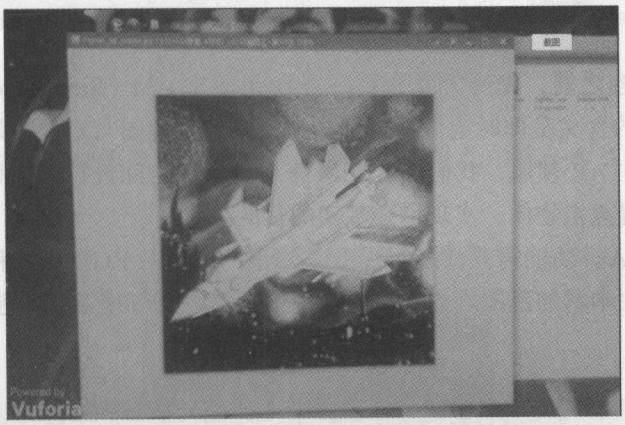
回到 Unity 中，添加一个截图的 Button，将 ScriptsGameObject 物体拖至它的 OnClick 事件中，并选定屏幕截图函数。

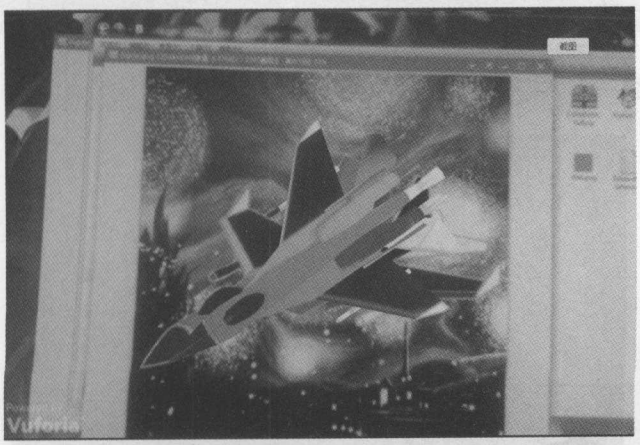


调整按钮的位置和大小以及显示的文字。



运行测试。





识别到识别图时，按下截图按钮就能正确显示模型颜色，至此，涂色类AR的核心功能已经实现了，我们可以在此功能的基础上进行扩展，做出自己的涂色类AR项目。

第8章 使用陀螺仪模拟 SLAM 功能

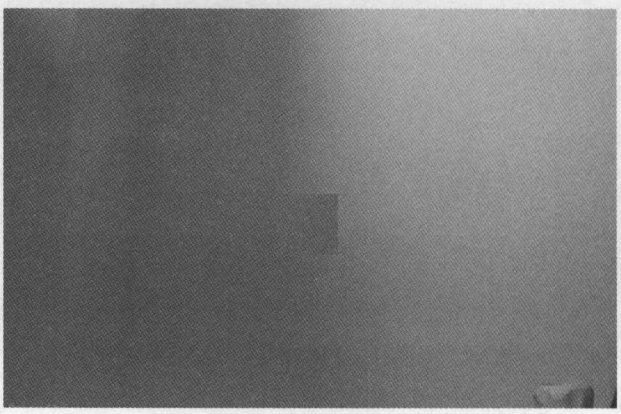
8.1 SLAM 的概念

在介绍本章内容之前，先给大家介绍一下什么是 SLAM（Simultaneous localization and mapping）。SLAM 的含义是即时定位与地图构建，或并发建图与定位，是实现真正全自主移动机器人的关键。在 AR 领域，我们用 SLAM 来对周围的空间进行扫描和三维空间重建，然后在重建的位置放置虚拟物体，实现增强现实的目的。

实现真正的 SLAM 功能需要涉及很多数学知识，以及对 RGB 图像和深度图像的处理等等，这些内容并不在本章的范围之内，有兴趣的读者可以自行搜索资源进行学习！

8.2 模拟 SLAM 功能的实现

首先给大家介绍一下本节最终实现的效果：

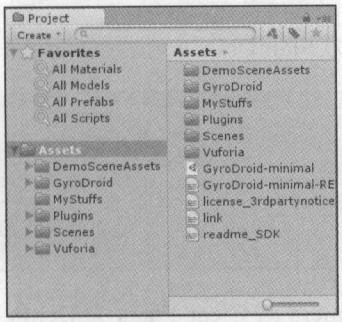


我们不借助识别图或者其他识别信息，直接在屏幕中看到一个绿色的 Box，当手机移开时，Box 会消失，再回来的时候，绿色的 Box 还会出现在这个位置。

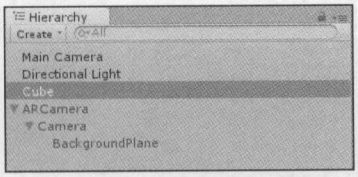
实现原理是：场景中放置两个 Camera，一个是 ARCamera，用来调用摄像头以显示真实世界，另一个 Camera 用来渲染虚拟物体，这里指的是绿色的 Box，然后用陀螺仪来控制第二个摄像机的旋转，这样就可以模拟 SLAM 的效果。

由于传感器的误差、波动等一系列因素的影响，如果直接使用而不进行滤波处理，效果势必会不稳定，所以我们采用的解决办法是引入第三方插件 [GyroDroid]。

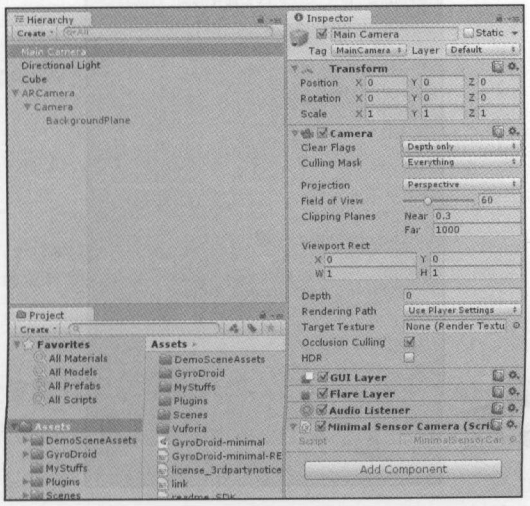
首先打开 Unity，创建一个新的工程，将 GyroDroid 和 Vuforia 的 SDK 导入该工程，再新建一个文件夹存放我们制作的内容。



在默认的场景中，不要删除默认的摄像机，直接在场景中拖一个 ARCamera，然后创建一个 Cube。

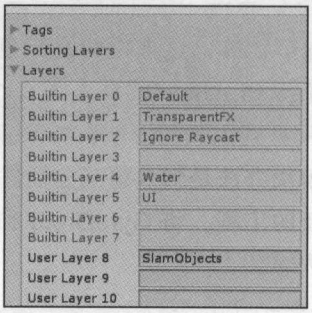


接下来，我们需要把 Main Camera 的 Clear Flags 属性改为 Depth only。

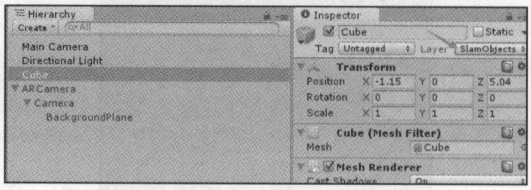


创建一个材质球，将颜色设置为绿色，并把该材质赋给 Cube。

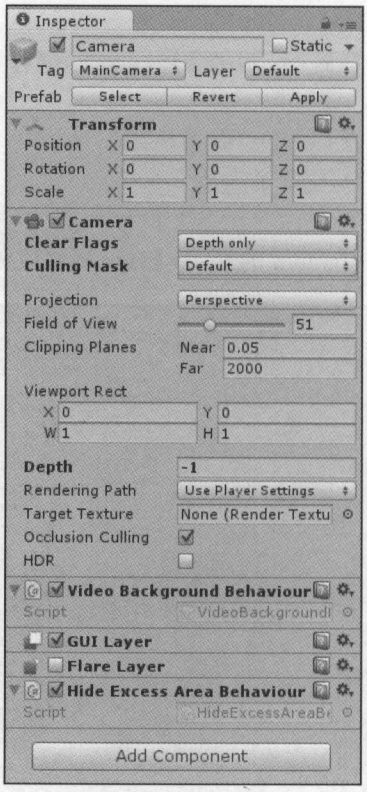
新建一个 Layer（层），命名为 SlamObjects。



然后将 Cube 的 Layer 设置为 SlamObjects。

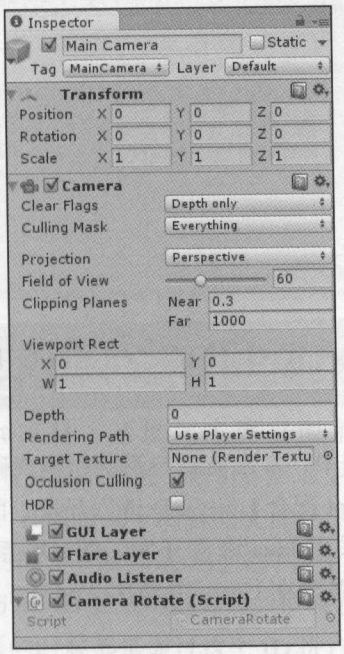


然后设置 ARCamera 下 Camera 的 Clear Flags 为 Depth only，CullingMask 的属性改为 Default，这样 ARCamera 将不会渲染绿色的 Cube。



这里我们用ARCamera只是为了调用外部摄像头，还可以用WebCamTexture来实现，大家可以自己研究。这里我们需要给ARCamera添加License Key，添加方法前面已经多次介绍过了，这里就不再重复。

接下来，编写代码实现用陀螺仪和加速计控制摄像机的旋转，新建一个CameraRotate的C#脚本，然后将它拖给MainCamera。



在该脚本中编写以下代码：

```
using UnityEngine;
using System.Collections;

public class CameraRotate : MonoBehaviour
{
    void Start()
    {
        SensorHelper.ActivateRotation();// 激活旋转传感器

    }

    void Update()
    {
        transform.rotation = SensorHelper.rotation;
        // 让该物体的旋转和手机陀螺仪的旋转一致
    }
}
```

代码非常简单，大家看注释即可。完成之后打包项目到Android平台上测试，实现模拟SLAM的功能！

第9章 Unity与原生代码之间的交互

9.1 Unity与Android原生代码之间的交互

在我们开发AR产品的过程中，常常需要与系统原生代码进行相互调用，因为目前大部分AR应用都是用C#、JavaScript配合Unity 3D进行开发，而大部分传统应用都是使用原生语言开发，比如Android系统采用Java语言开发，而iOS系统采用Objective-C或Swift进行开发。这样我们不可避免地需要在Unity中调用Android原生接口，或从Objective-C给Unity接口发送消息。

本节我们重点讲述Unity与Android原生代码之间的相互调用。要实现这个功能，我们必须要对Android系统开发和Unity开发都有比较透彻的了解。相信阅读本书的读者对于Unity的掌握程度会比较好，但是对Android开发并不一定很了解，因此我们首先来简要介绍一下Android开发的基础知识。

1. 开发环境

Android开发环境主要是Eclipse和Android Studio，本节内容主要是在Eclipse下进行讲述，如果你还没有在Eclipse下配置好Android开发环境，请先按照以下步骤下载、安装和配置环境。

（1）下载并安装JDK（Java Development Kit），JDK是Java开发工具包，因为Android系统是用Java进行编写的，所以我们需要确保Java的正常运行。JDK安装完成后，需要配置环境变量，环境变量的配置过程比较简单，只需要在谷歌或百度上搜索步骤即可。

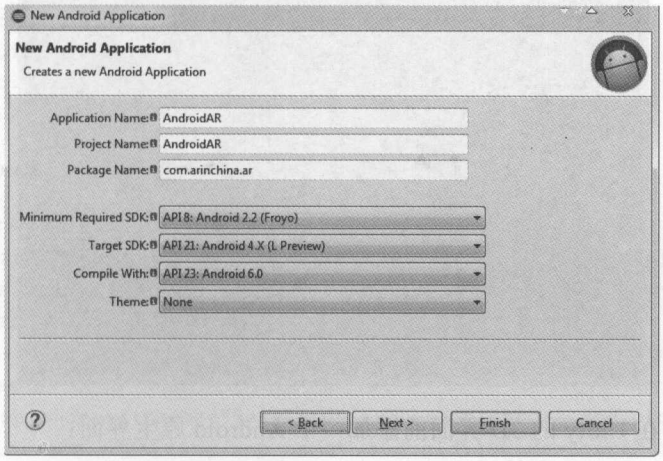
（2）下载并安装Android SDK（Android Software Development Kit），Android SDK是指Android软件开发工具包，从谷歌开发者官网获得后进行安装，并在SDK Manager工具中选择Android某一版本进行下载安装。

（3）在Eclipse中配置并指定Android SDK。

2. 工程结构

开发环境配置完成后，我们可以新建一个Android工程，下面将通过这个工程讲解

Android 的工程结构。新建工程如下图所示。



在新建的工程中主要有以下文件目录结构：

（1）src：所有逻辑和功能相关的 Java 文件。

（2）gen：自动生成的 Java 文件，这里面主要有两个 Java 文件，分别为 R.java 和 BuildConfig.java，前者的功能是对资源文件进行索引，后者主要是配置文件。

（3）assets：用于存放资源，在该目录下的资源文件并没有在 R.java 中生成索引，访问时需要用 AssetsManager 类进行访问，一般都是通过文件路径进行访问操作。

（4）res：用于存放资源，与 assets 目录不同的是，所有在 res 下的资源都会在 R.java 中生成对应的 ID，在访问这些资源的时候都是通过这个 ID 进行访问的，因此如果某个资源没有用到，该资源是不会打包进程序最终的文件里的，而 assets 目录下的资源则会全部打包进程序文件。

（5）libs：该目录主要存放项目引用到的类库以及第三方库。

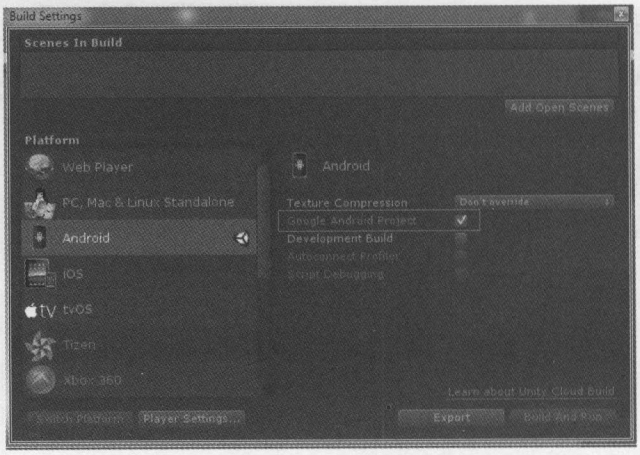
（6）AndroidManifest.xml 文件：该文件是当前工程的总配置文件，在该文件中定义了 SDK 版本号、所有的 Activity 以及该应用会调用系统的哪些特性等一系列配置需求。

3. 将 Unity 工程导出为 Android 工程

一般情况下，我们是直接在 Unity 中导出 Android 安装文件，即 apk 文件，但是要做到 Unity 的 App 与 Android 原生 App 之间的融合，必须要将 Unity 工程导出为 Android 工程。导出 Android 工程的方法比较简单，只需要在 Build Settings 中选中 Google Android Project 选项，然后点击 Export 即可导出 Android 工程。

4. 实现原生界面与 Unity 界面交互功能

用 Eclipse 打开从 Unity 导出的工程，我们可以看到该工程中有 UnityPlayerActivity.java 这个文件，这就是我们进行 Unity 与 Android 互通的关键所在，其实 Unity 的界面都是由 UnityPlayerActivity 进行管理的，而 UnityPlayerActivity 这个类本身是继承自 Activity 类的。Activity 是 Android 系统的五大组件之一，通常称为活动，其管理的目标可以认为是界面，从一个界面切换到另外一个界面都是由 Activity 之间的切换来实现的。



接下来，我们在 Unity 的 3D 界面前添加一个 Android 原生界面：

（1）创建新的 Activity 文件。

```
//原生界面
//MainMenuActivity.java
public class MainMenuActivity extends Activity
{

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        this.setContentView(R.layout.mainmenu);

        //查询界面上的按钮
        Button btn = (Button)this.findViewById(R.id.startButton);
        //给按钮添加点击事件，并让它切换到 Unity 的 Acitivity
        btn.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent it = new Intent();
                it.setClass(MainMenuActivity.this, UnityPlayerActivity.class);
                MainMenuActivity.this.startActivity(it);
            }
        });
    }

}
```

（2）创建原生界面布局文件 mainmenu.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

```
    <Button android:id = "@+id/startButton"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "@string/start_button"
        android:clickable="true"/>
</LinearLayout>
```

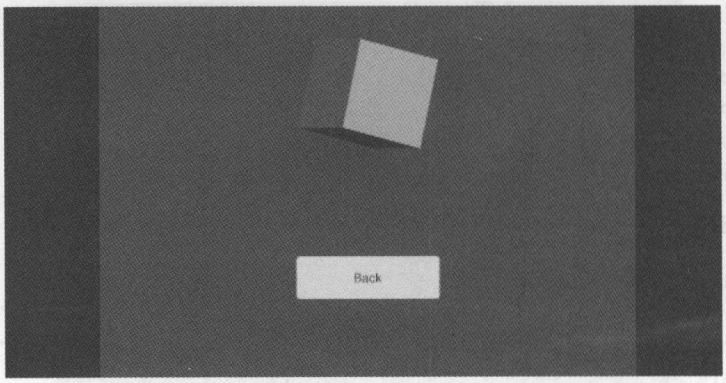
（3）在AndroidManifest文件中设置原生Activity为启动界面。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.Move.AU" android:versionName="1.0" android:versionCode="1" android:insta
llLocation="preferExternal">
  <supports-screens android:smallScreens="true" android:normalScreens="true"
android:largeScreens="true" android:xlargeScreens="true" android:anyDensity="true" />
  <application android:theme="@style/UnityThemeSelector" android:icon="@drawable/
app_icon" android:label="@string/app_name" android:isGame="true" android:banner="@
drawable/app_banner">
    <activity android:label="@string/app_name" android:screenOrientation="fullSen
sor" android:launchMode="singleTask" android:configChanges="mcc|mnc|locale|touchscreen
|keyboard|keyboardHidden|navigation|orientation|screenLayout|uiMode|screenSize|smalle
stScreenSize|fontScale" android:name="com.Move.AU.UnityPlayerActivity">
      <meta-data android:name="unityplayer.UnityActivity" android:value="true" />
    </activity>
    <activity android:label="@string/app_name" android:screenOrientation="ful
lSensor" android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboardHidden|n
avigation|orientation|screenLayout|uiMode|screenSize|smallestScreenSize|fontScale"
android:name="com.Move.AU.MainMenuActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
        <category android:name="android.intent.category.LEANBACK_LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="23" />
  <uses-feature android:glEsVersion="0x00020000" />
  <uses-feature android:name="android.hardware.touchscreen" android:required="false" />
  <uses-feature android:name="android.hardware.touchscreen.multitouch" android:
required="false" />
  <uses-feature android:name="android.hardware.touchscreen.multitouch.distinct"
android:required="false" />
</manifest>
```

在上面的xml文件中可以看到，我们将主Acitivity和启动界面设置为MainMenuActivity，而不是原本的UnityPlayerActivity。

通过上述步骤，我们可以让App的启动界面为Android原生界面，然后跳转到Unity界面，接下来，我们再实现从Unity界面跳转至Android原生界面的功能。

（4）在Unity中创建场景，并添加一个按钮。



（5）创建 Game.cs 脚本，并添加以下代码：

```
using UnityEngine;
using System.Collections;

public class Game : MonoBehaviour
{
    //点击返回按钮时调用
    public void OnBackButtonClick()
    {
        using (AndroidJavaClass jc = new AndroidJavaClass("com.unity3d.player.
UnityPlayer"))
        {
            using (AndroidJavaObject jo = jc.GetStatic<AndroidJavaObject>("curren
tActivity"))
            {
                //调用 Android 插件中 UnityPlayerActivity 的 StartActivity0 方法
                //stringToEdit 表示它的参数
                jo.Call("BackToMenu", "msg");
            }
        }
    }
}
```

（6）在 UnityPlayerActivity 中添加 BackToMenu 函数，在该函数中调用切换界面功能。

```
public void BackToMenu(String msg)
{
    this.runOnUiThread(new Runnable()
    {
        public void run()
        {
            //切换到主界面功能
            Intent it = new Intent();
            it.setClass(UnityPlayerActivity.this, MainMenuActivity.class);
            UnityPlayerActivity.this.startActivity(it);
        }
    });
}
```

上述步骤完成后，我们就可以实现 Android 原生代码与 Unity 代码之间的交互功能。

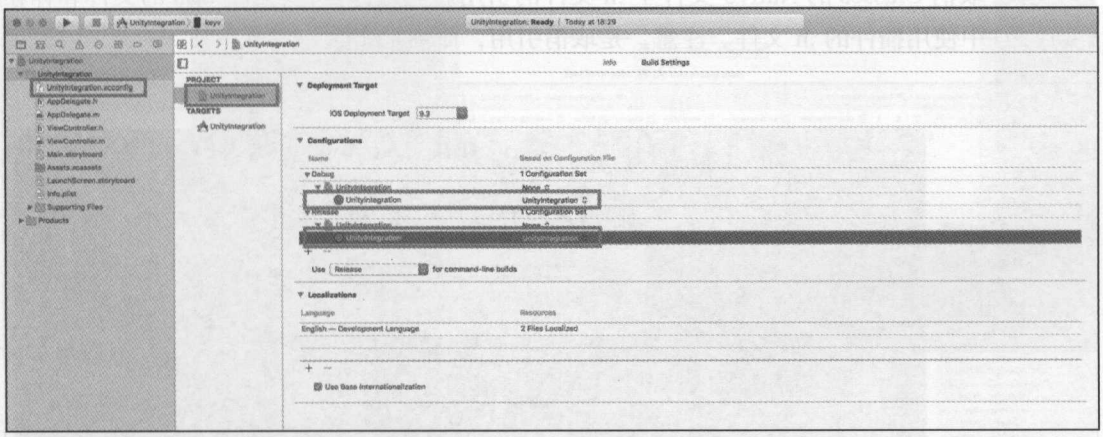
9.2 Unity 与 iOS 原生代码之间的交互

在实际的软件开发中，往往会遇到这样的问题：在一个用原生代码开发的 iOS 应用中需要加入 AR 的场景功能。我们不可能在 Unity 中重新做一遍，这既没有必要也不容易实现，所以我们要在 iOS 原生代码开发的工程中调用 Unity 中制作好的 AR 场景，即实现 Unity 与 iOS 原生代码之间的相互调用。

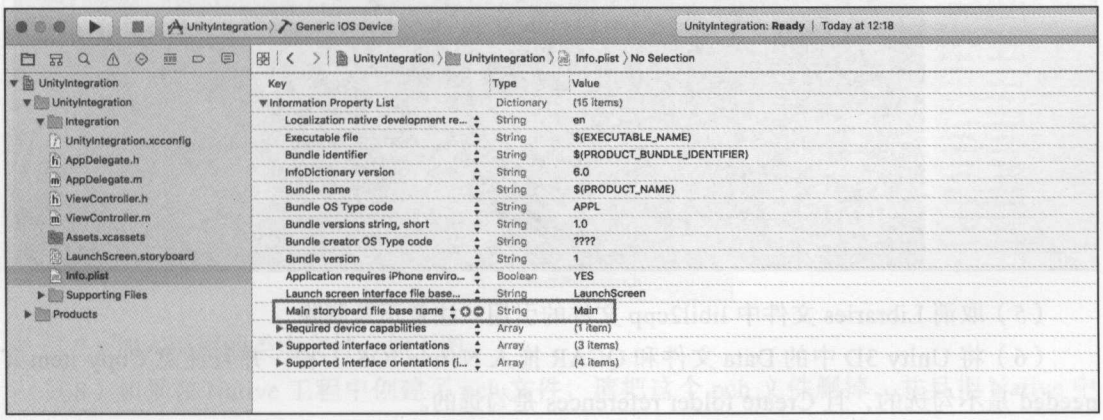
我们需要在 iOS 原生的工程中加入一个 Unity 3D 的 Vuforia 工程，基本的流程如下：首先在导出 Unity 3D 工程中设置 il2cpp，支持 Target iOS Version 8.0，然后根据从 Unity 3D 导出的 Xcode 工程，设置 Native Xcode 工程，接下来将 Unity 的工程代码文件导入 Native，最后在 main.mm 方法中改变程序入口，来回切换 Unity 3D 视图和原生视图。

（1）在 Native Xcode 工程中创建一个 UnityIntergration.xcconfig，以便后面合成的时候使用。

选择 project's setting，然后选择 info → configurations。

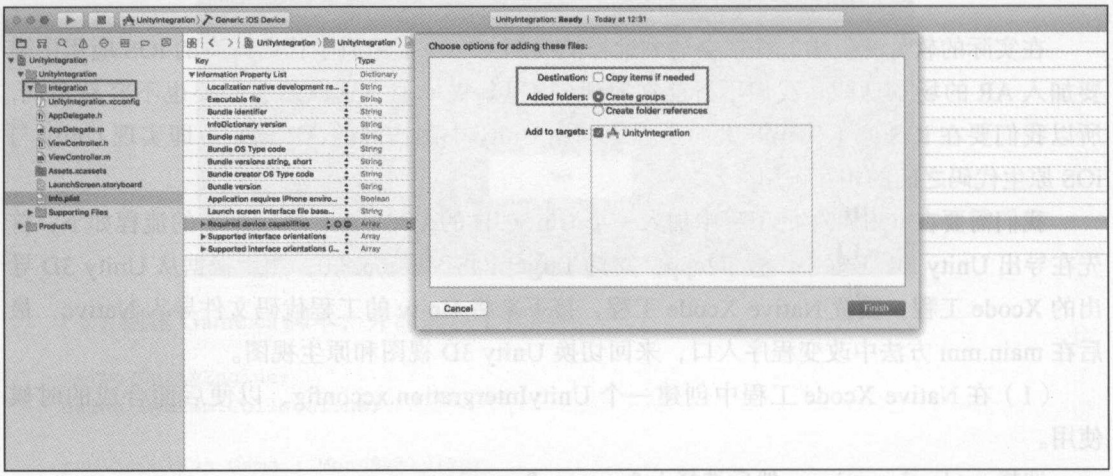


（2）去掉 info.plist 文件中的 Main.storyboard。

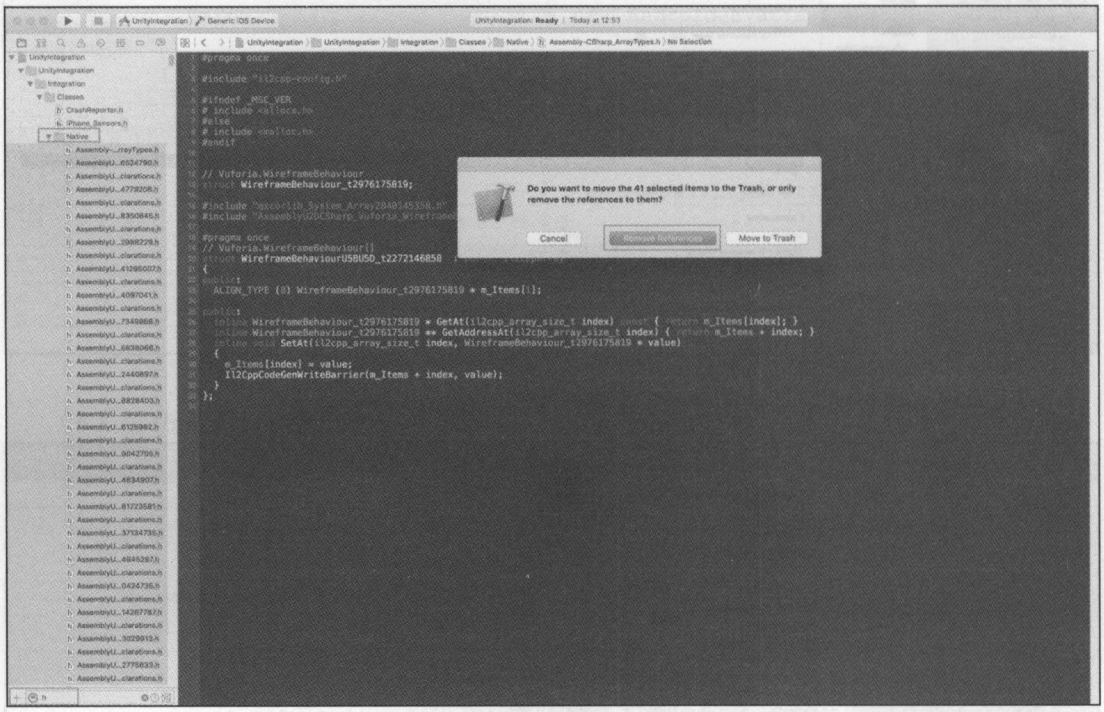


（3）在原生工程中创建一个用来放置 Unity 3D 工程的文件夹，然后拖入从 Unity 3D 导

出的 Xcode 工程到 Native iOS 创建的文件夹中，在导入的时候确保是 Classes 和 Libraries 两个文件夹。在导入的时候确保 Copy item if needed 是不勾选的，且 Create group 是勾选的。

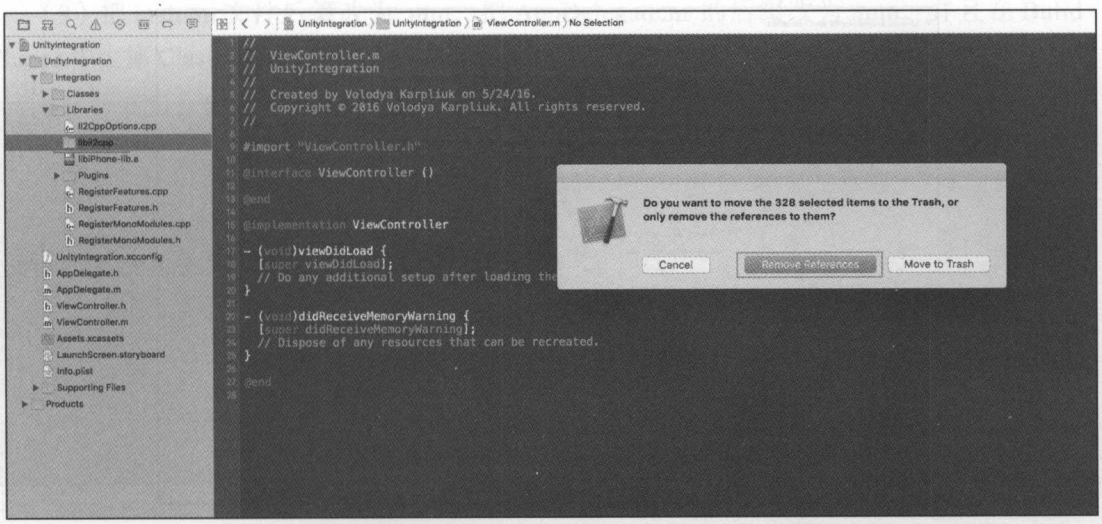


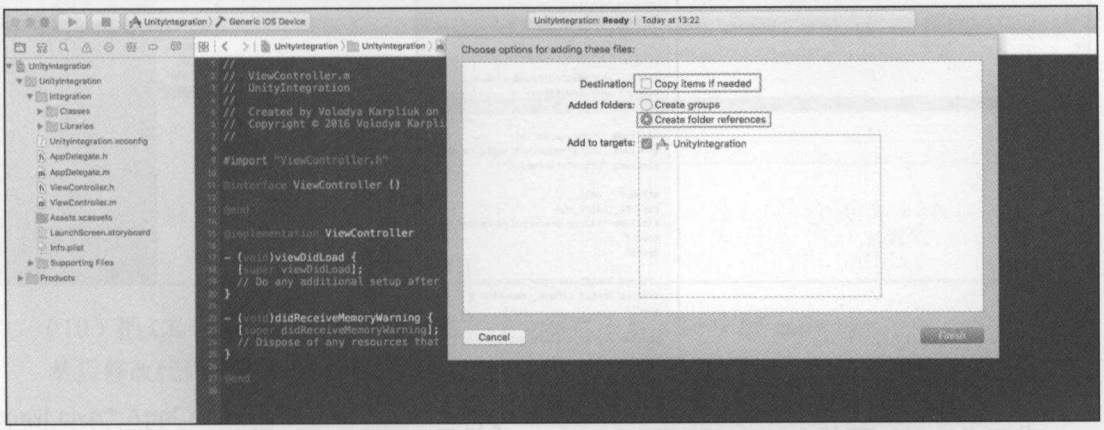
（4）取消 Classes 的 Native 文件中 .h 文件的引用。但是保留包含 Vuforia 的文件和你在 Unity 3D 中使用插件的 .h 文件。注意，是取消引用，而不是删除。

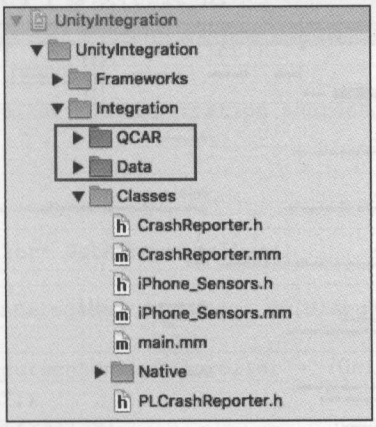


（5）取消 Libraries 文件中 libil2cpp 文件的引用，注意不是删除。

（6）将 Unity 3D 中的 Data 文件和 QCAR 拖入 Native iOS 工程。并且注意 Copy item if needed 是不勾选的，且 Create folder references 是勾选的。

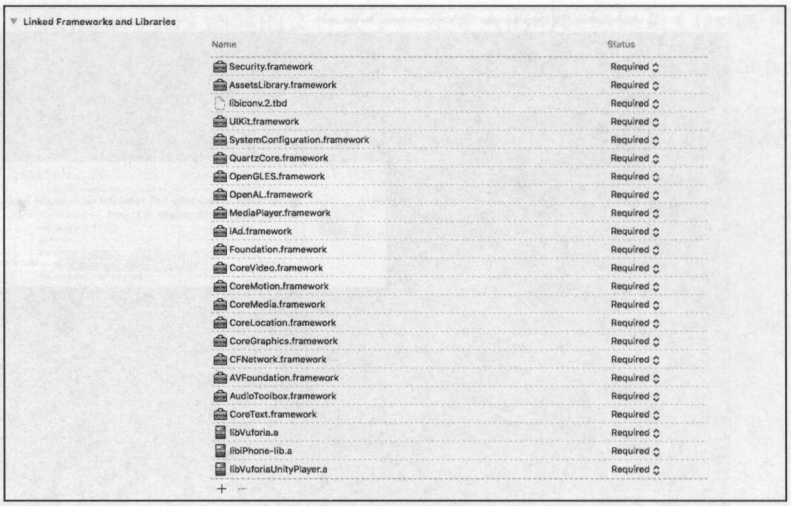






（7）加入引用的类库文件，如下图。

（8）如果在 Native 工程中创建了 pch 文件，请把这个 pch 文件删掉，并且把 Native 中的 pch 文件引用内容拷入 Unity 3D 生成的 pch 文件中。




```
#ifndef PrefixHeader_pch
#define PrefixHeader_pch

#ifdef __OBJC__
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#endif

#include "Preprocessor.h"
#include "UnityTrampolineConfigure.h"
#include "UnityInterface.h"

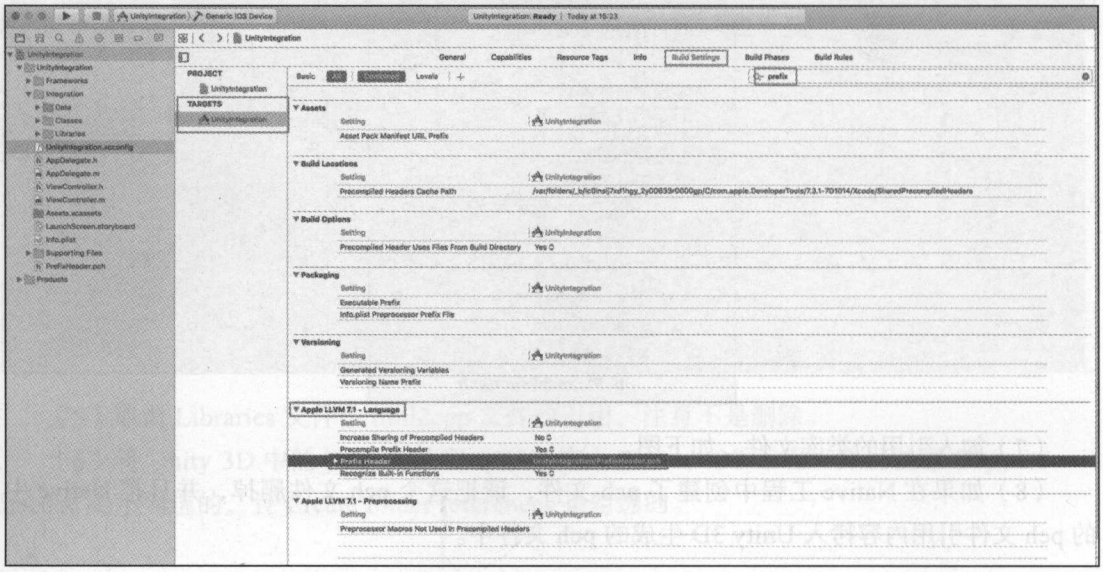
#ifndef __OBJC__
#if USE_IL2CPP_PCH
#include "il2cpp_precompiled_header.h"
#endif
#endif

#ifndef TARGET_IPHONE_SIMULATOR
#define TARGET_IPHONE_SIMULATOR 0
#endif

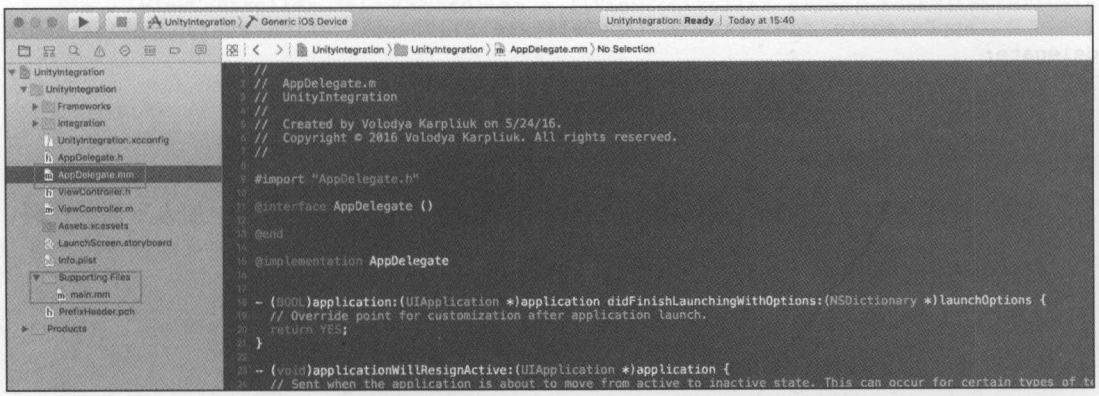
#define printf_console printf

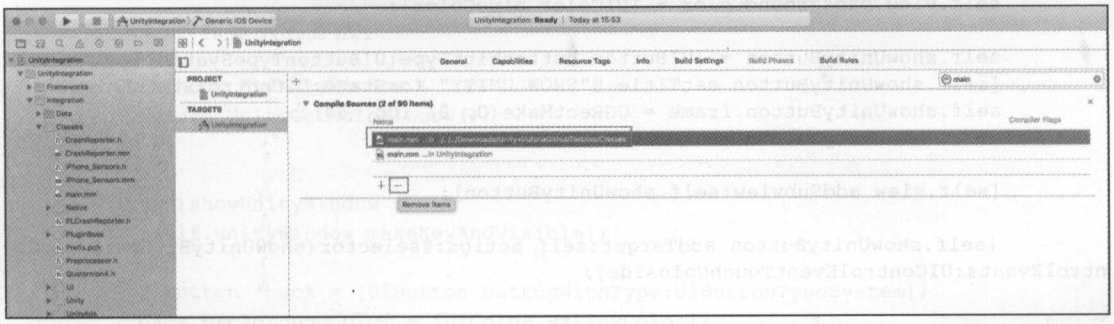
#endif
```

在 Buildsetting 下的 Apple LLVM7.1-Language 中设置 Prefix Header 的 pch 文件引用路径。



（9）把 main.m 文件后缀改为 .mm，把 AppDelegate.m 的后缀改为 .mm，并且在 Bulid Phases 中去掉 Classes 中的一个 main.mm。





（10）把 Classes/main.mm 中的代码文件全部拷入 Native 中的 main.mm。

然后修改代码，把 const char* AppControllerClassName = “UnityAppController”修改为 const char* AppControllerClassName = “AppDelegate”。

（11）在 UnityAppController.h 中修改代码，把

```
inline UnityAppController*  GetAppController()
{
    return (UnityAppController*)[UIApplication sharedApplication].delegate;
}
```

修改为

```
NS_INLINE UnityAppController* GetAppController()
{
    NSObject<UIApplicationDelegate>* delegate = [UIApplication sharedApplication].
delegate;
     UnityAppController* currentUnityController = (UnityAppController *)[delegate
valueForKey:@"unityController"];
    return currentUnityController;
}
```

（12）在 UnityAppController.mm 文件中修改代码，导入 #import “AppDelegate.h”，把

```
- (void)shouldAttachRenderDelegate{}
```

修改为

```
- (void)shouldAttachRenderDelegate  {

        AppDelegate *delegate = (AppDelegate *)[UIApplication sharedApplication].
delegate;

    [delegate shouldAttachRenderDelegate];

}
```

（13）首先创建一个 ViewController.h，然后在 ViewController.m 文件中加入下面的代码：

```
#import "AppDelegate.h"

@property (nonatomic, strong) UIButton *showUnityButton;

- (void)viewDidLoad {
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor blueColor];

    self.showUnityButton = [UIButton buttonWithType:UIButtonTypeSystem];
    [self.showUnityButton setTitle:@"SHOW UNITY" forState:UIControlStateNormal];
    self.showUnityButton.frame = CGRectMake(0, 0, 100, 44);
    self.showUnityButton.center = self.view.center;

    [self.view addSubview:self.showUnityButton];

    [self.showUnityButton addTarget:self action:@selector(showUnityButton:) forCo
ntrolEvents:UIControlEventTouchUpInside];

}

- (void)showUnityButton: (UIButton *) sender {
  [(AppDelegate *)[UIApplication sharedApplication].delegate showUnityWindow];
}
```

（14）我们来显示 Unity 界面和原生界面的来回跳转，在 AppDelegate.h 中加入

```
@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) UIWindow *unityWindow;

@property (strong, nonatomic) UnityAppController *unityController;

- (void)showUnityWindow;
- (void)hideUnityWindow;

- (void)shouldAttachRenderDelegate;
```

（15）在 AppDelegate.mm 文件中导入 #import “ ViewController.h” 和 #import “ VuforiaRender Delegate.h”，添加一个导航控制器和属性。

```
@property (nonatomic, strong) UINavigationController *navVC;
extern "C" void VuforiaRenderEvent(int marker);
```

随后在 didFinishLaunchingWithOptions 中加入代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(N
SDictionary *)launchOptions {
```

```
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
    self.window.backgroundColor = [UIColor redColor];

    ViewController *viewController = [[ViewController alloc] initWithNibName:nil
bundle:nil];

    self.navVC = [[UINavigationController alloc] initWithRootViewController:viewC
ontroller];
    self.window.rootViewController = self.navVC;

    self.unityController = [[UnityAppController alloc] init];
    [self.unityController application:application didFinishLaunchingWithOptions:l
aunchOptions];

    [self.window makeKeyAndVisible];

    return YES;
}
```

实现隐藏和显示的方法：

```
- (UIWindow *)unityWindow {
    return UnityGetMainWindow();
}

- (void)showUnityWindow {
    [self.unityWindow makeKeyAndVisible];

    UIButton *back = [UIButton buttonWithType:UIButtonTypeSystem];
    back.backgroundColor = [UIColor yellowColor];
    [back setTitle:@"BACK" forState:UIControlStateNormal];
    back.frame = CGRectMake(0, 0, 100, 44);
    back.center = self.unityWindow.center;

    [self.unityWindow addSubview:back];

    [back addTarget:self action:@selector(hideUnityWindow) forControlEvents:UICon
trolEventTouchUpInside];


}

- (void)hideUnityWindow {
    [self.window makeKeyAndVisible];
}
```

重写 shouldAttachRenderDelegate：

```
- (void)shouldAttachRenderDelegate {

    self.unityController.renderDelegate = [[VuforiaRenderDelegate alloc] init];

    UnityRegisterRenderingPlugin(NULL, &VuforiaRenderEvent);

#if UNITY_VERSION>434
```

```
    UnityRegisterRenderingPlugin(NULL, &VuforiaRenderEvent);

#endif

}
```

最后重写以下状态的方法：

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state。
This can occur for certain types of temporary interruptions (such as an incoming
phone call or SMS message) or when the user quits the application and it begins the
transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down
OpenGL ES frame rates。Games should use this method to pause the game.
    [self.unityController applicationWillResignActive:application];
}

- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate
timers, and store enough application state information to restore your application to
its current state in case it is terminated later.
    // If your application supports background execution, this method is called
instead of applicationWillTerminate: when the user quits.
    [self.unityController applicationDidEnterBackground:application];
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive
state; here you can undo many of the changes made on entering the background.
    [self.unityController applicationWillEnterForeground:application];
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the
application was inactive。If the application was previously in the background,
optionally refresh the user interface.
    [self.unityController applicationDidBecomeActive:application];
}

- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate。Save data if
appropriate。See also applicationDidEnterBackground:.
    [self.unityController applicationWillTerminate:application];
}
```

至此所有的 iOS 和 unity 3D AR 场景的跳转代码结合完毕，可以运行了。此后由于版本不同或者设置不同，或许还会碰到问题，但是不会再出现大问题了，我们可以自行解决。

最后，有兴趣的读者可参考以下链接：

- https://the-nerd.be/2015/11/13/integrate-unity-5-in-a-native-iOS-app-with-xcode-7/
- http://www.makethegame.net/unity/add-unity3d-to-native-iOS-app-with-unity-5-and-vuforia-4-x/
- https://github.com/blitzagency/iOS-unity5

第10章 使用 HiAR SDK 开发 AR 应用

10.1 搭建开发环境

要使用 HiAR SDK for Unity 进行开发，请安装下列软件并进行相关配置。

1. 下载安装 Unity

按照兼容性要求，前往官网下载并安装。同时，我们建议你注册 Unity 账户，以便于更好地使用 Unity 开发环境。

Unity 下载：https://store.unity.com/cn。

2. 下载安装 QuickTime（v1.1.x 及以后版本无需安装 QuickTime）

由于 Unity 平台限制，在 Windows 系统上，需要通过安装 QuickTime 才能更好地使用 HiAR SDK 中提供的视频 AR 功能。请务必安装！

QuickTime 下载：https://support.apple.com/zh-cn/HT201175。

3. 下载安装 Android SDK 和 JDK

安装 Android SDK 和 JDK 之后，Unity 可以导出 Android 工程和应用，并支持直接在 Android 设备上进行调试运行。

下载 Android SDK：https://developer.android.com/studio/index.html?hl=zh-cn。

下载 JDK ：http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html。

注意

如果无法正常下载，请尝试通过 VPN 进行访问。

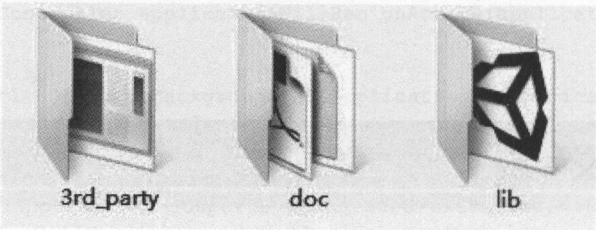
完成开发环境的搭建后便可以开始创建 Unity 工程并导入 SDK。

10.2 导入 SDK

下面介绍如何在 Unity 工程中导入 HiAR SDK for Unity。在开始之前，请先访问 HiAR 官网下载最新版本的 SDK。

1. 下载并解压 SDK

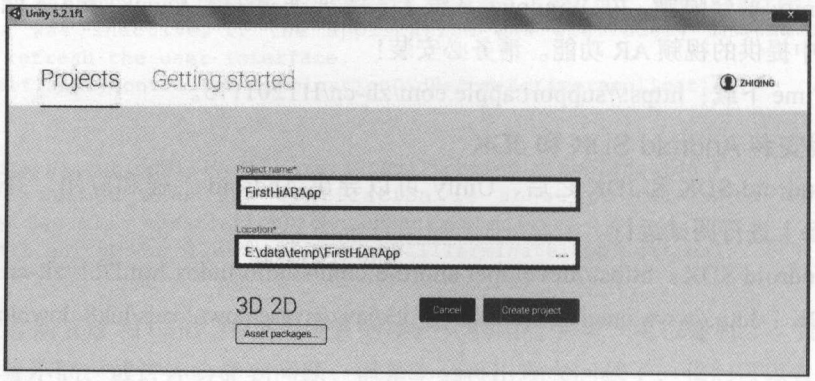
SDK 的下载链接为 http://hiar.com.cn/download.html，前往下载页面下载 SDK 包，可以获得一个 zip 文件：hiar_sdk_unity_yyyymmdd_vx.x.x_xxxxxxx.zip（文件名随版本更新而变化）。将 zip 文件解压，可看到如下目录结构：



- lib 目录：包含 Unity 插件文件 hiar_sdk_unity-x.x.x.unitypackage。
- doc 目录：包含使用说明文档、license.txt 和 changelog.txt。
- 3rd_party 目录：包含第三方库文件（1.0.6 之后的版本有该目录）。

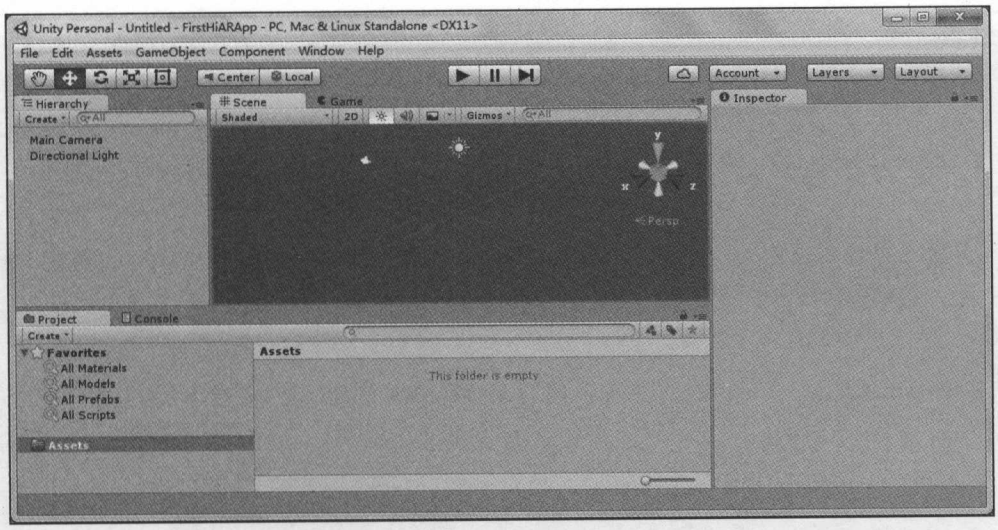
2. 新建 Unity 工程

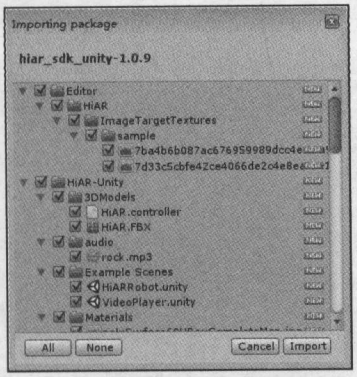
运行 Unity 程序，并按照步骤新建一个 Unity 工程。当然，也可以直接打开已存在的工程来进行导入工作。



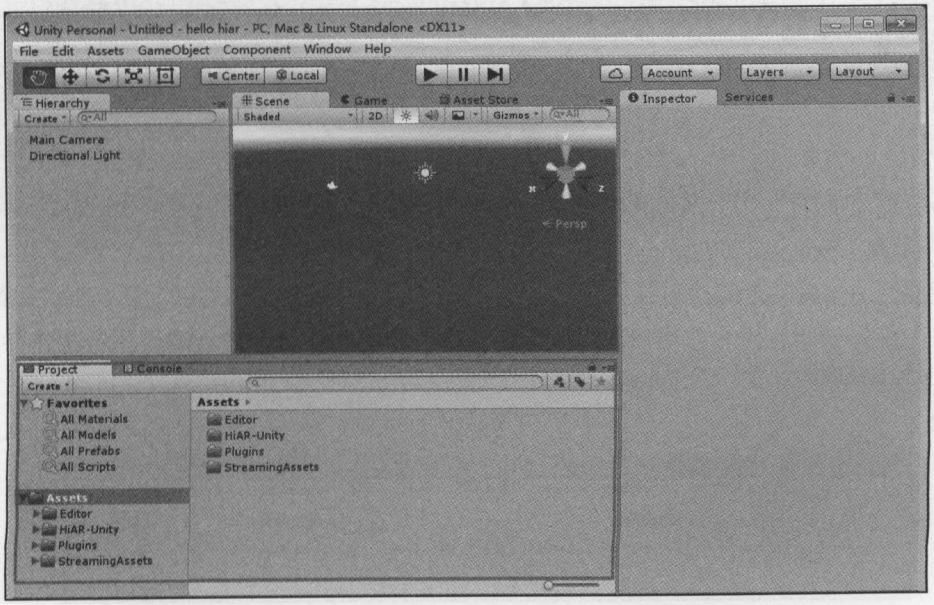
3. 导入 HiAR SDK for Unity

双击 hiar_sdk_unity-x.x.x.unitypackage 文件，Unity 将加载并显示插件内容，点击 Import 开始导入。也可以在 Unity 中点击 Assets → Import Package → Custom Package... 来导入插件。





导入完成后可以在 Project 窗口中看到以下内容。



注意

Unity 插件与一般意义上的插件存在区别，每次在新工程中使用时都需要重新进行导入。

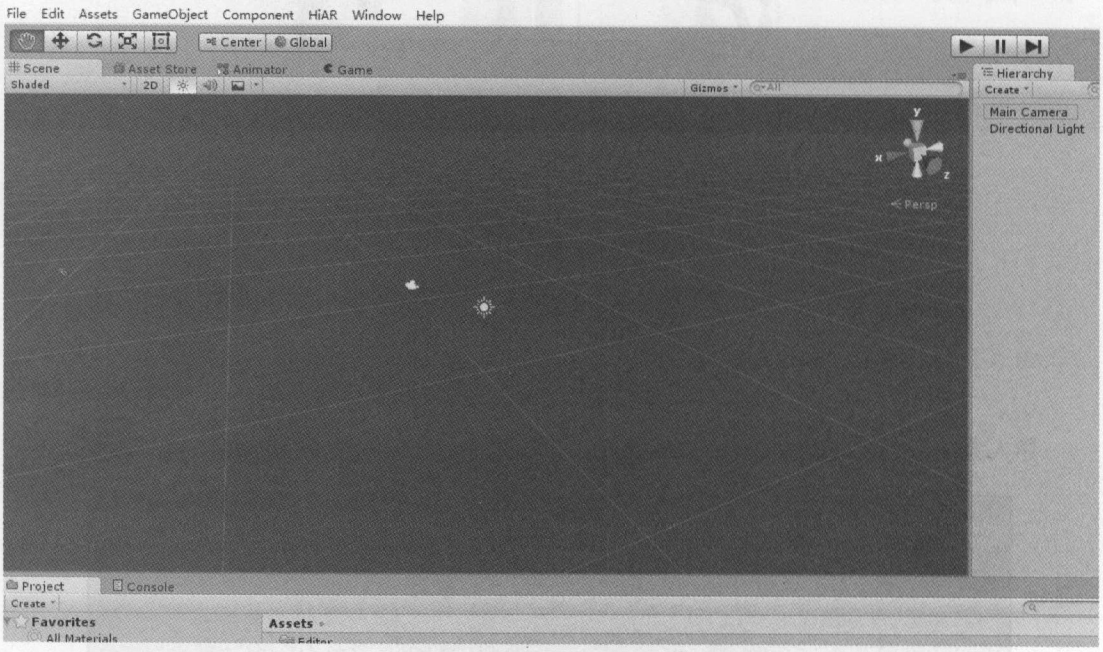
至此，SDK 导入工作已经完成，可以开始创建 HelloWorld。

10.3 创建 HelloWorld

本节将介绍如何在 Windows 系统下使用 HiAR SDK 创建一个简单的 AR 应用。

1. 创建 HiARCamera

在新建的 Unity 工程中会默认创建一个 Main Camera（如下图），请先将其删除。



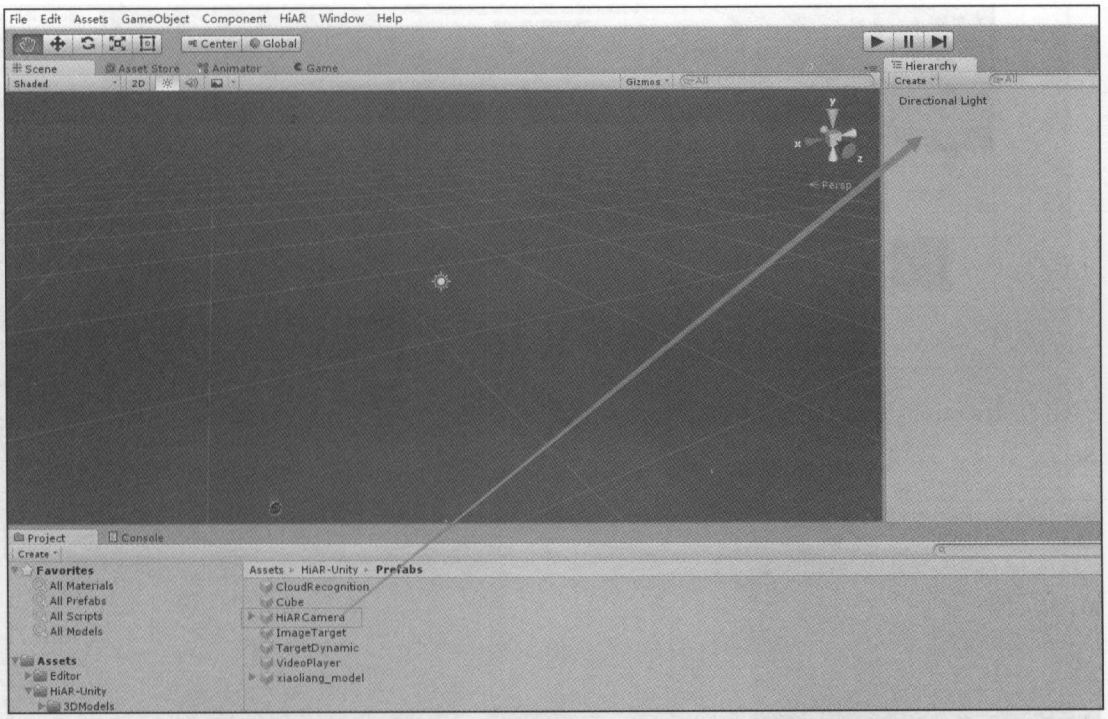
然后将 Assets/HiAR-Unity/Prefabs 目录下的 HiARCamera 拖至 Hierarchy 窗口。

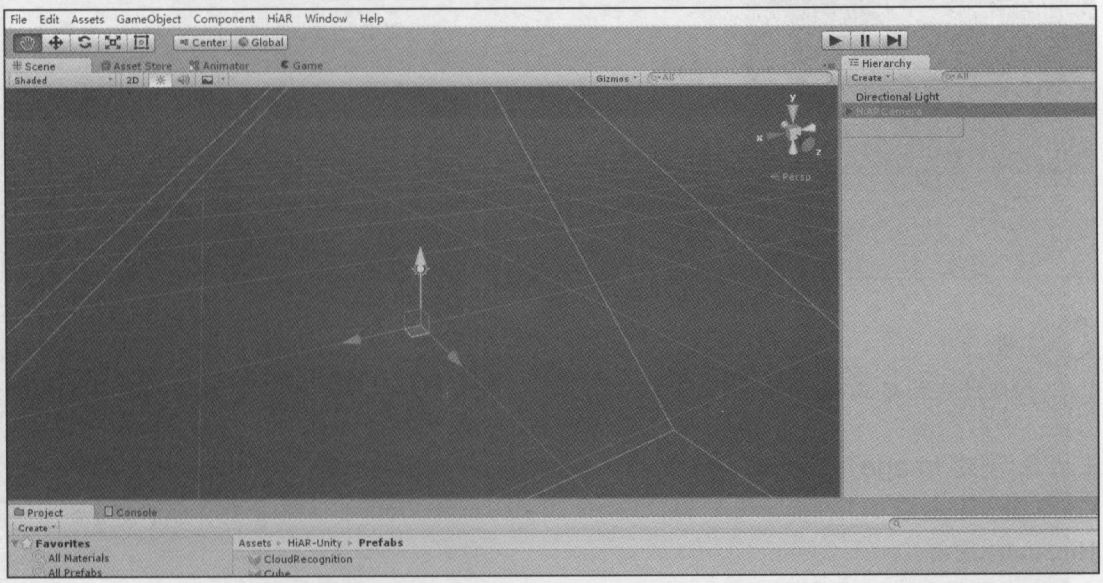
2. 设置 AppKey 和 Secret

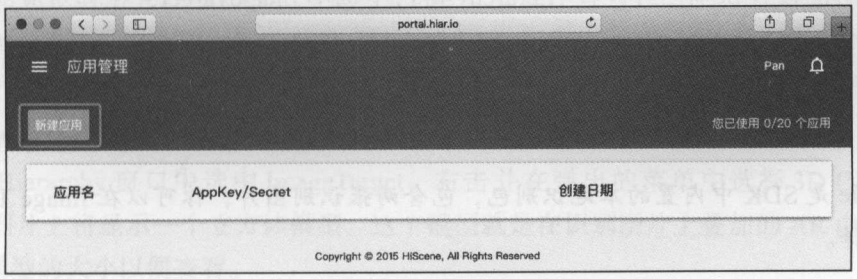
要使用 HiAR 平台的功能，必须创建应用并获取 AppKey 和 Secret。

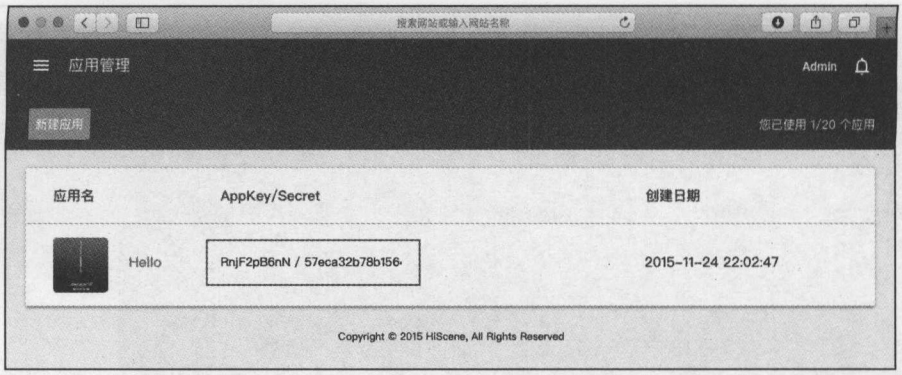
（1）首先创建应用。前往 HiAR 进行后台的应用管理，点击“新建应用”按钮并按照提示要求创建一个新应用。

（2）下面查看应用信息。系统会为每个应用分配唯一的 AppKey 和 Secret。进入应用管理界面，可以在列表中查看应用对应的 AppKey 和 Secret。

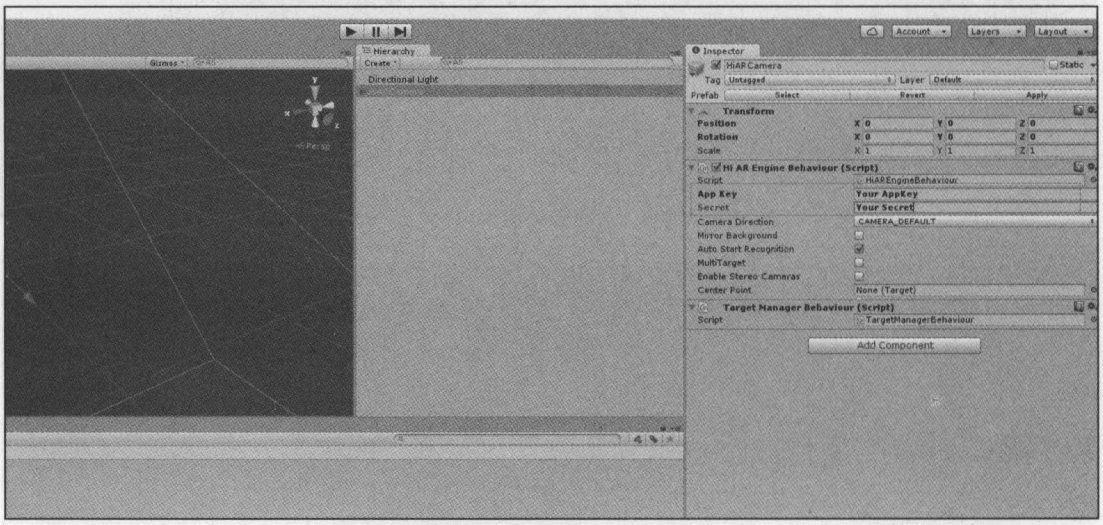

File Edit Assets GameObject Component HiAR Window Help
Center Global
Scene
Asset Store
Animator
Game
Shaded
2D
Gizmos
Hierarchy
Create
Directional Light
Persp
Project
Console
Create
Favorites
All Materials
All Prefabs
All Scripts
All Models
Assets
Editor
HiAR-Unity
3DModels
Assets ▸ HiAR-Unity ▸ Prefabs
CloudRecognition
Cube
HiARCamera
ImageTarget
TargetDynamic
VideoPlayer
xiaoliang_model


File Edit Assets GameObject Component HiAR Window Help
Center Global
Scene
Asset Store
Animator
Game
Shaded
2D
Gizmos
Hierarchy
Create
Directional Light
HiARCamera
Persp
Project
Console
Create
Favorites
All Materials
All Prefabs
Assets ▸ HiAR-Unity ▸ Prefabs
CloudRecognition
Cube


portal.hiar.io
应用管理
Pan
新建应用
您已使用 0/20 个应用
应用名
AppKey/Secret
创建日期
Copyright © 2015 HiScene, All Rights Reserved



在 Hierarchy 窗口中选中 HiARCamera，在其对应的 Inspector 窗口中找到 AppKey 和 Secret 选项，并填写信息。



注意

如果没有指定正确的 AppKey 和 Secret，应用可能无法正常运行。

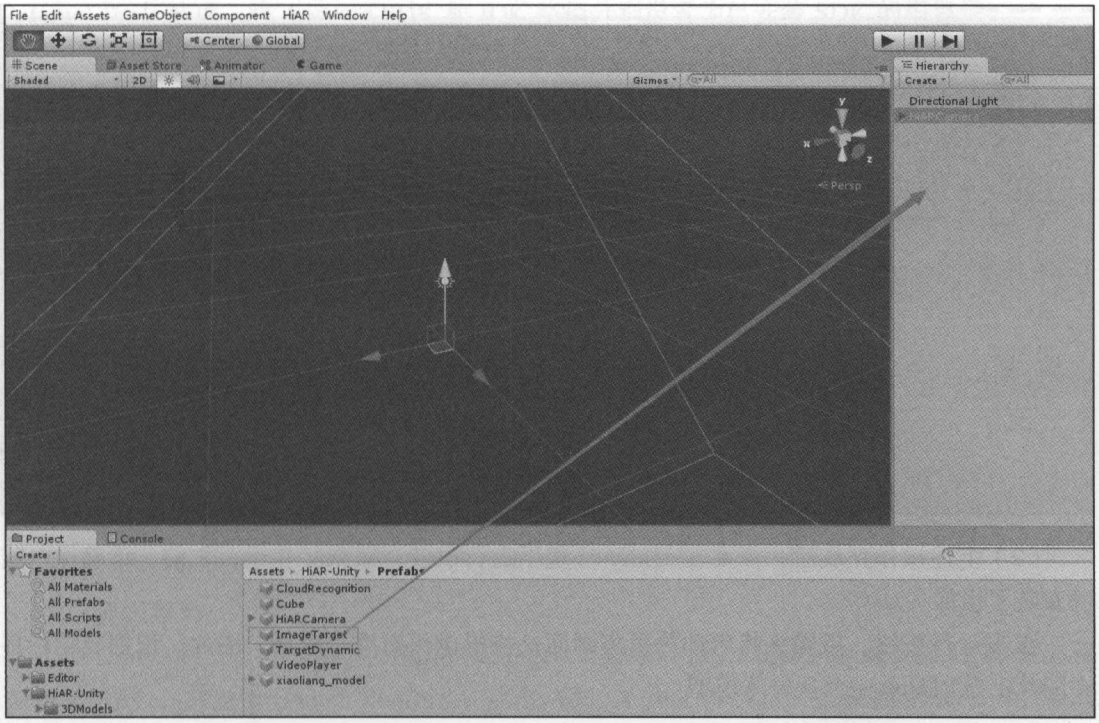
3. 创建 ImageTarget

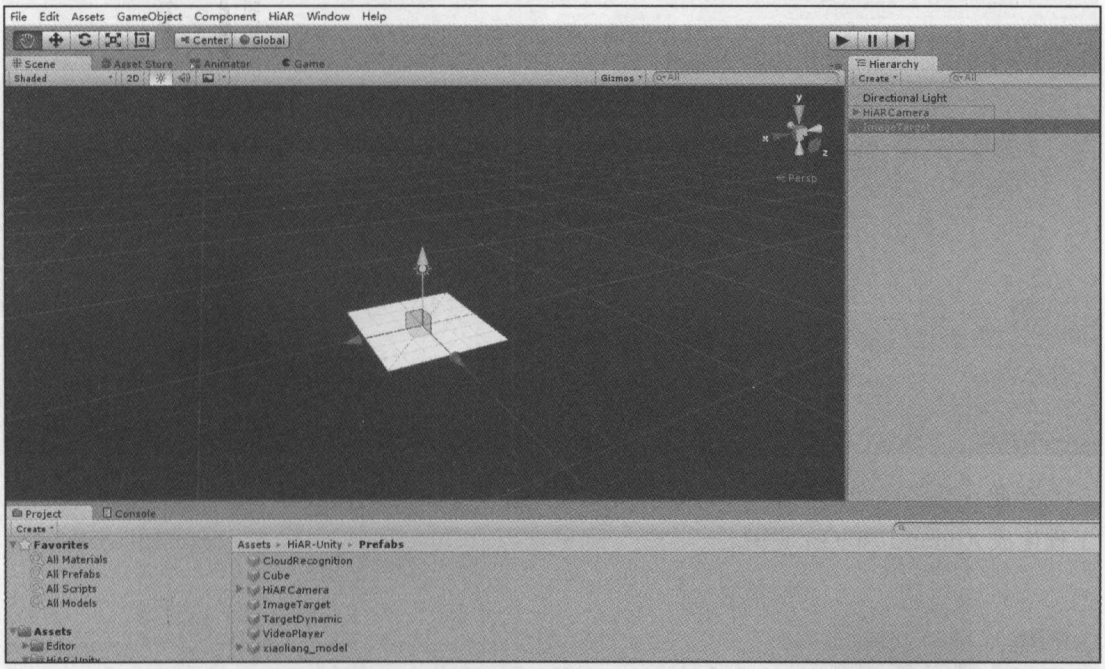
（1）首先添加 ImageTarget，将 Assets/HiAR-Unity/Prefabs 目录下的 ImageTarget 拖至 Hierarchy 窗口。

（2）下面设置识别图片，在 Hierarchy 窗口中选中 ImageTarget，在其对应的 Inspector 窗口中找到 HiAR Target Mono Behaviour（Script），在 Data Set 选项中选择 sample。此时，编辑窗口中将显示一张“苹果”的图片，这是刚刚设置的识别图片。

注意

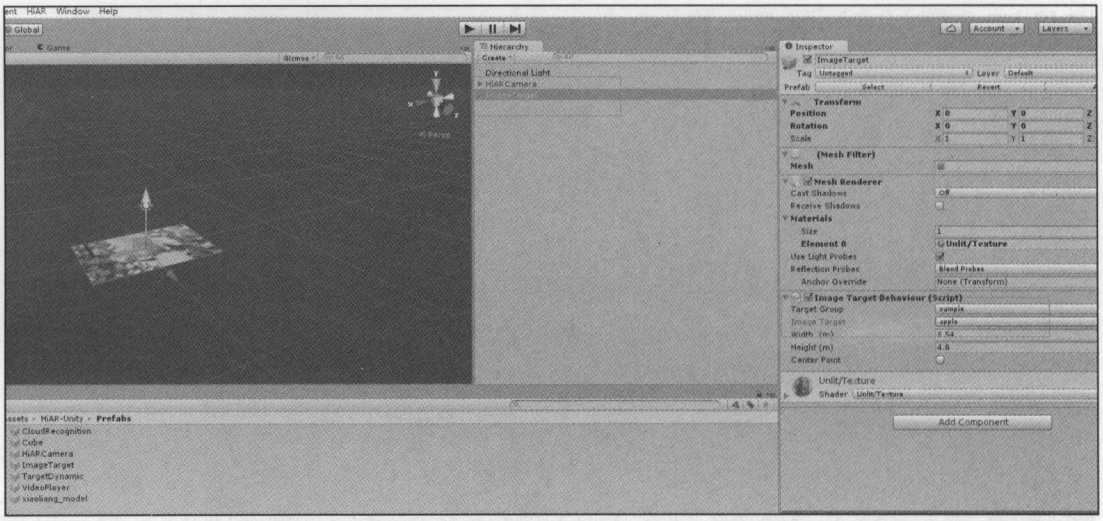
sample 是 SDK 中内置的本地识别包，包含两张识别图片，你可以在 Image Target 选项中进行选择。





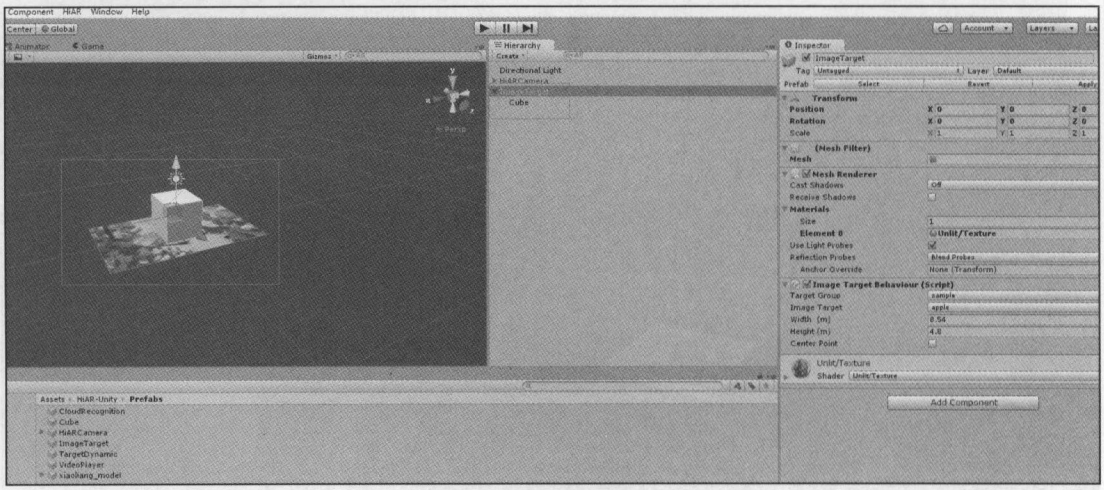
4. AR 制作与运行

（1）在 Hierarchy 窗口中选中 ImageTarget，右击并在弹出的菜单中选择 3D Object → Cube，识别图片上将显示一个立方体模型。这个模型就是在识别图片上叠加的 AR 内容，可以适当调整模型的大小以便查看。



（2）上述操作完成后，需要在 Unity 环境下运行以查看效果。在运行之前，请确保电脑已安装了摄像头。

点击运行按钮，摄像头将启动并采集画面；将摄像头对准“苹果”图片，识别成功后将在图片上叠加显示一个立方体模型。



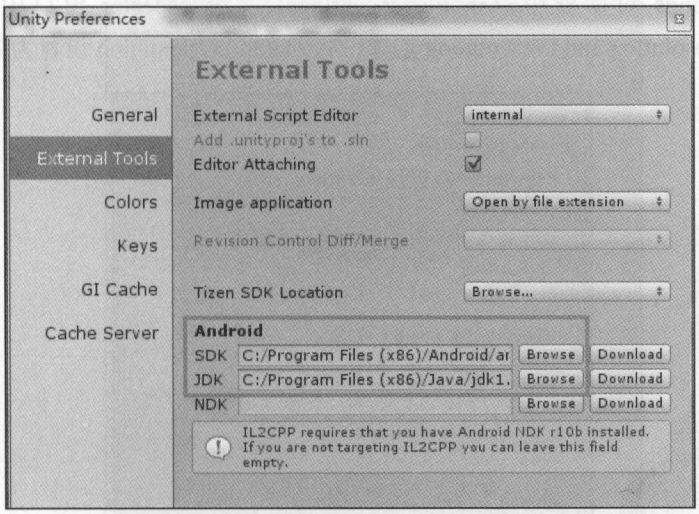
至此，你已成功创建了一个简易的 AR 应用，可以继续制作更多 AR 效果，或者将当前工程生成 Android 和 iOS 应用。

10.4 导出 Android 工程

1. 设置 Android SDK 和 JDK 路径

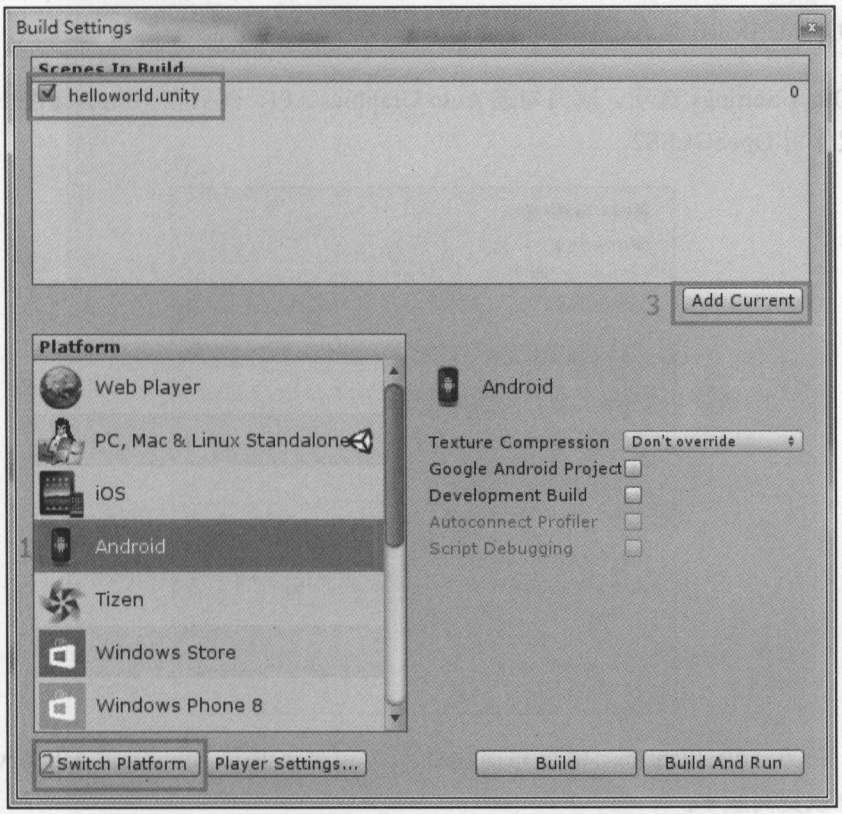
在 Unity 中，点击 Edit → Preferences...，在弹出的 Unity Preferences 对话框中点击

External Tools 选项，将 Android SDK 和 JDK 路径分别设置为已安装 SDK 的根目录。



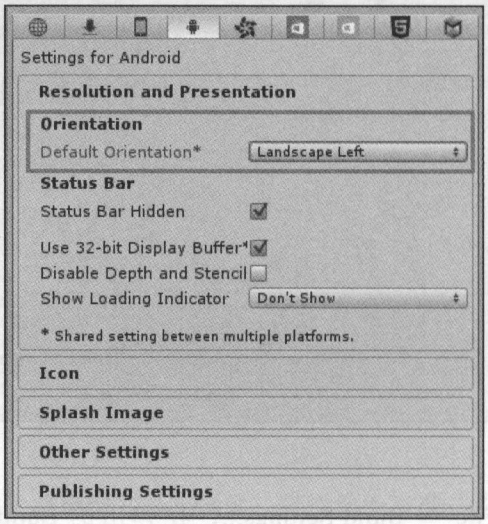
2. 选择平台

在 Unity 中，点击 File → Build Settings...，在弹出的 Build Settings 对话框中选择 Android 平台，并点击 Switch Platform，点击 Add Current 将当前工程添加到 Scene In Build 窗口中（请确保已被勾选）。



3. 应用设置

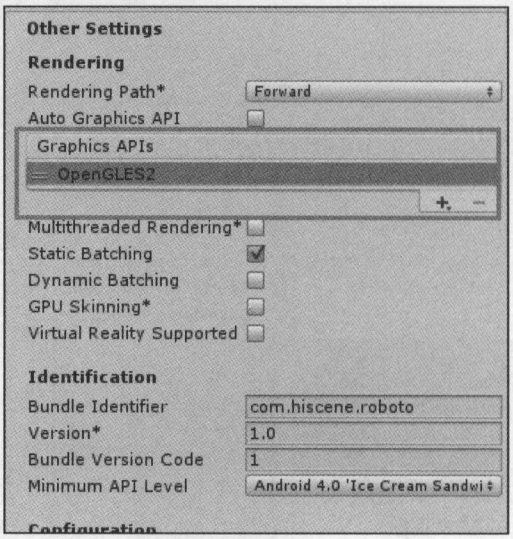
在 Build Settings 对话框中点击 Player Settings...，在 Inspector 窗口中找到 Settings for Android，点击 Resolution and Presentation 选项，将 Default Orientation 设置为 Landscape Left。



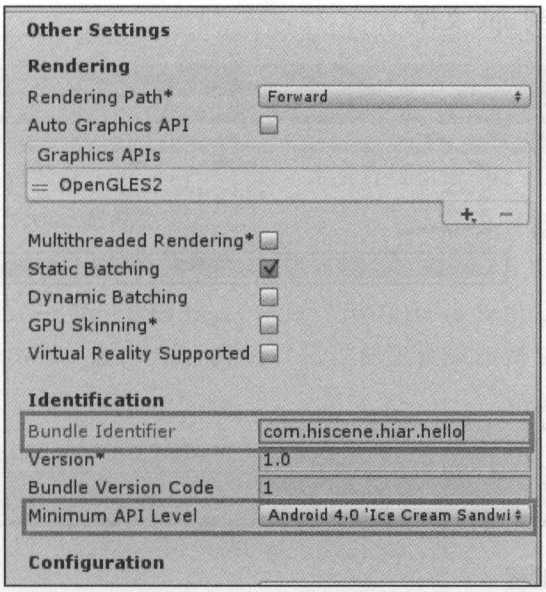
注意：

生成的 Android 应用暂时不支持 Auto Rotation，另外，有的手机由于自身系统原因不支持 Portrait Upside Down 模式。

点击 Other Settings 选项，取消勾选 Auto Graphics API，然后在 Graphics APIs 处删除多余选项，仅保留 OpenGLES2。

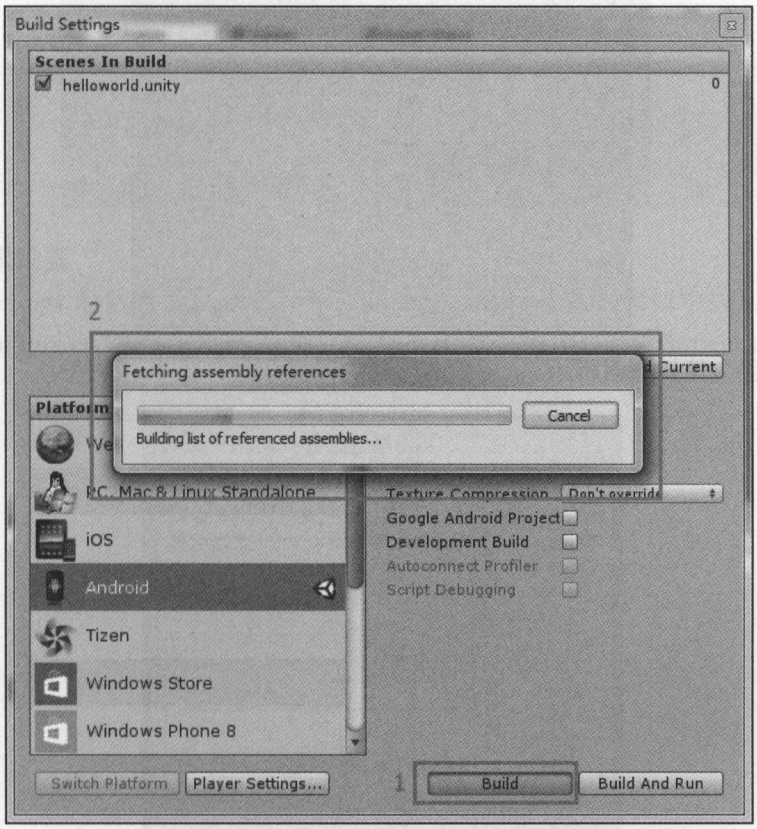


然后，在 Bundle Identifier 中填写应用的 apk 包名，并将 Minimum API Level 设置为 Android 4.0 或更高版本。

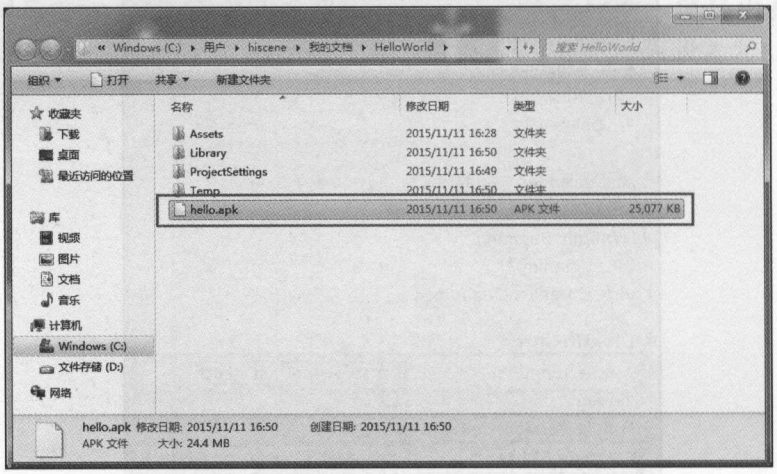


4. 生成 Android 应用

完成上述操作后，在 Build Settings 对话框中点击 Build，保存 apk 并开始生成应用。在此过程中，可能会要求你更新 Android SDK 和 JDK，请按照提示进行更新。

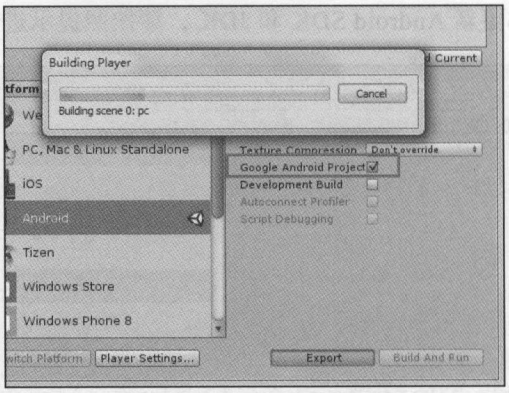


导出成功后可以看到 apk 文件。

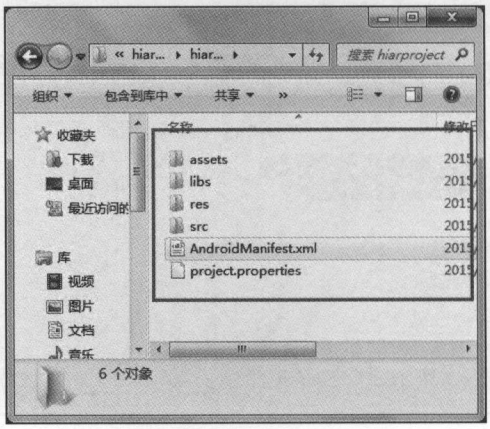


5. 导出 Android 工程

如果希望将当前工程导出为 Android 工程文件，则需要在第 4 步中勾选 Build Settings 对话框中的 Google Android Project，然后点击 Export。



导出的工程文件如下图所示。

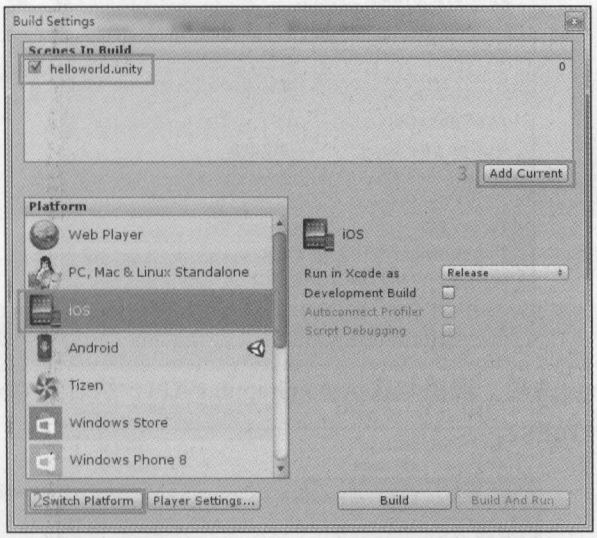


10.5 导出iOS工程

在开始之前，请务必先保存工程。由于Unity无法直接生成ipa文件，因此需要先导出iOS工程文件，然后通过Xcode编译生成。

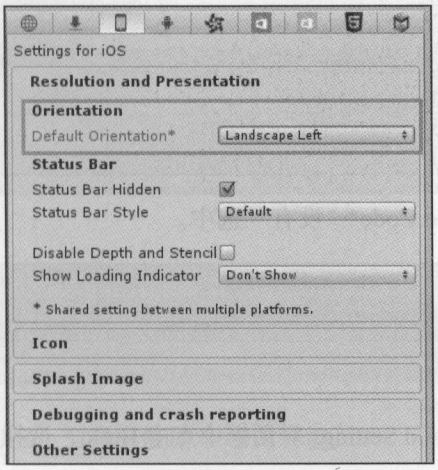
1. 选择平台

在Unity中，点击File → Build Settings...，在弹出的Build Settings对话框中选择iOS平台，并点击Switch Platform，点击Add Current将当前工程添加到Scene In Build窗口中（请确认已勾选）。



2. iOS应用设置

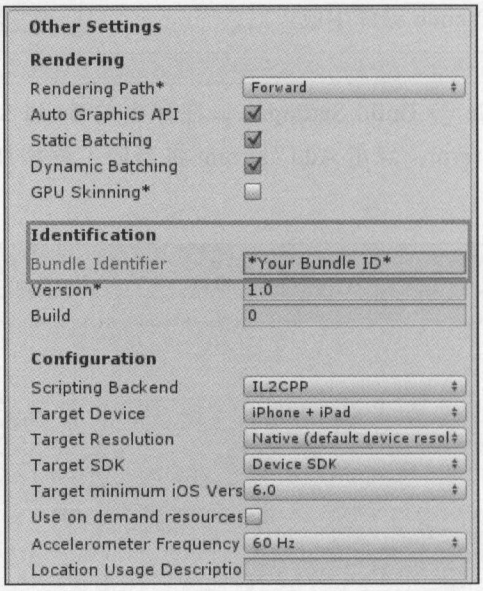
在Build Settings对话框中点击Player Settings...，在Inspector窗口中找到Settings for iOS，展开Resolution and Presentation选项，将Default Orientation设置为Landscape Left。



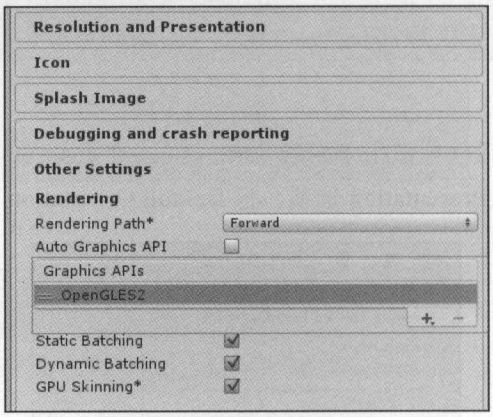
注意：

生成的 iOS 应用暂时不支持 Auto Rotation。

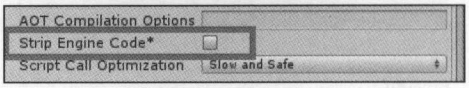
点击 Other Settings 选项，在 Bundle Identifier 中填写应用的 Bundle ID。



点击 Other Settings 选项，取消勾选 Auto Graphics API，然后在 Graphics APIs 处删除多余选项，仅保留 OpenGLES2。

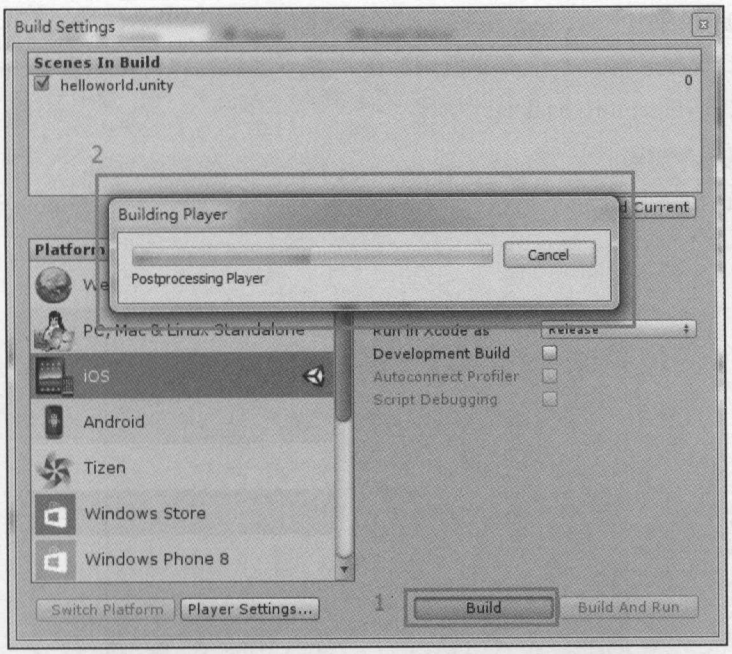


同时，确保 Strip Engine Code* “没有”选中。

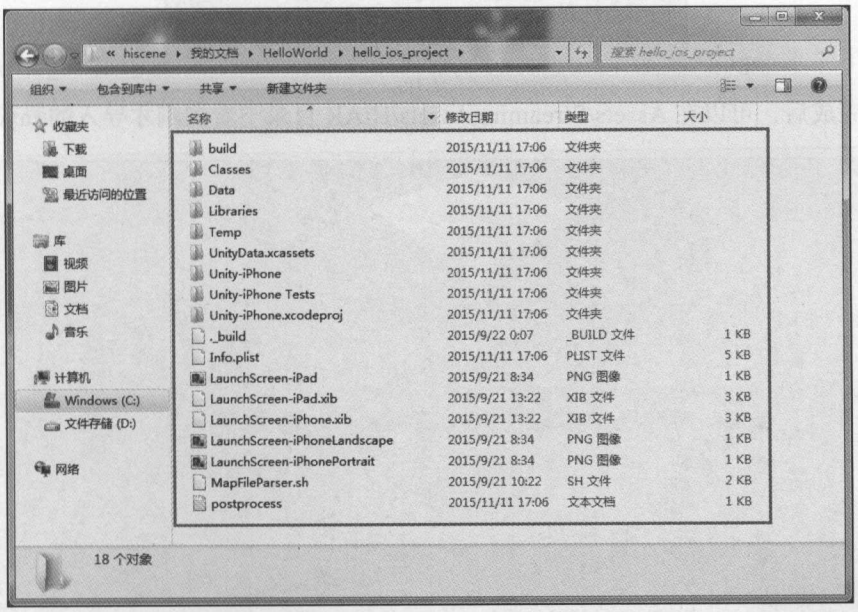


3. 导出 iOS 工程

完成上述操作后，在 Build Settings 对话框中点击 Build，选择目录并开始导出 iOS 工程。



导出成功后会得到 iOS 工程文件，可以通过 Xcode 进行编译来生成 iOS 应用。



10.6 使用本地识别包

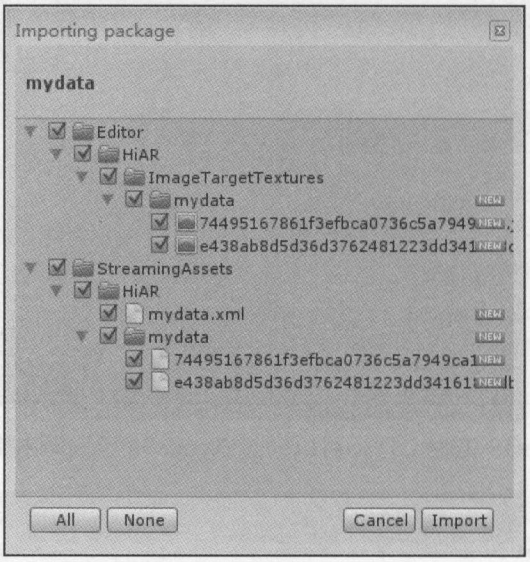
本节将介绍如何在 Unity 工程中使用本地识别包。

1. 下载本地识别包

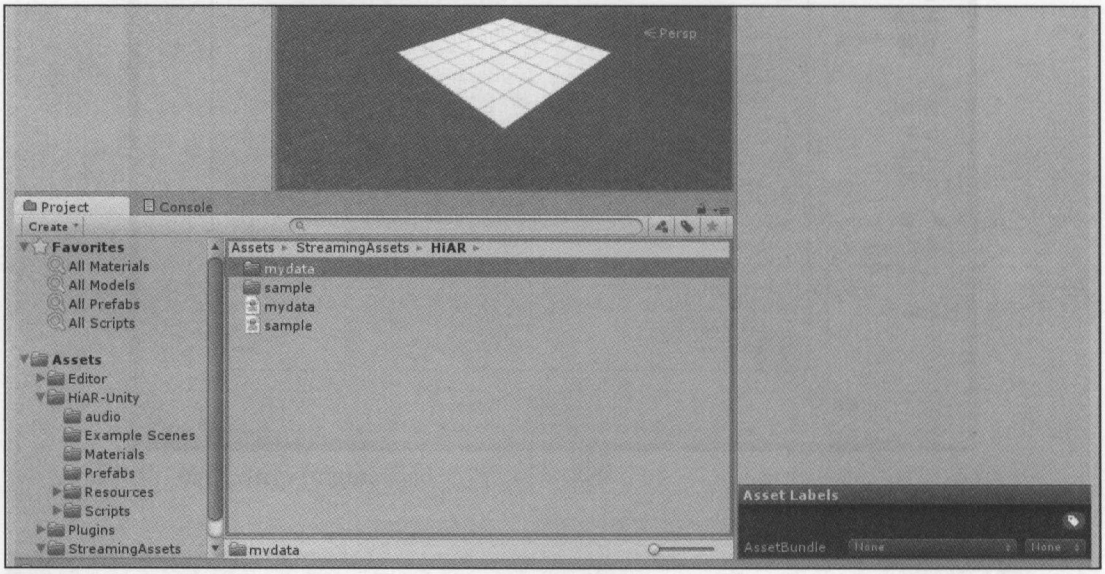
前往 HiAR 管理后台，上传需要识别的图片并下载识别包，可以获得一个 unitypackage 文件（本例中以 mydata.unitypackage 为例）。

2. 导入本地识别包

双击 mydata.unitypackage 文件，Unity 加载并显示包内容，点击 Import 开始导入。

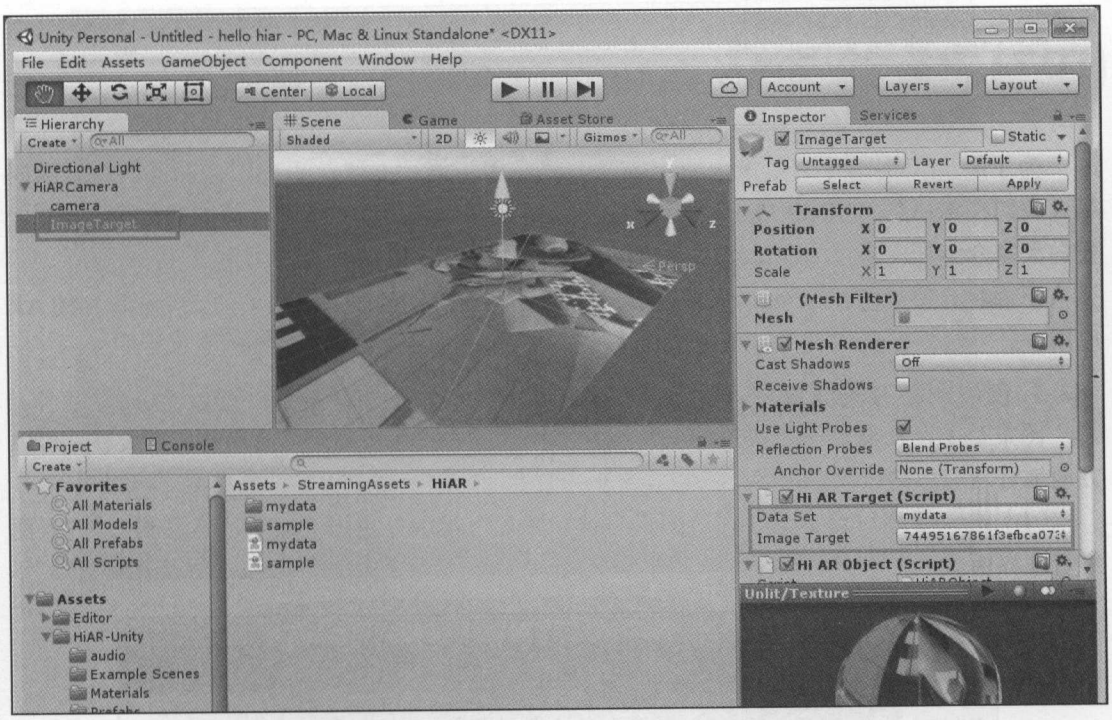


导入完成后，可以在 Assets/StreamingAssets/HiAR 目录下看到刚才导入的 mydata。



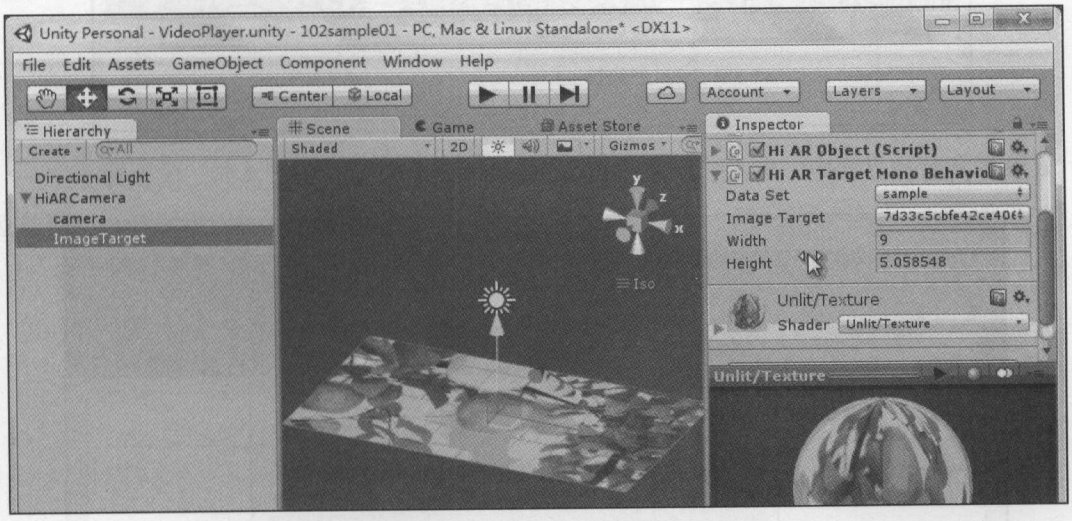
3. 设置 ImageTarget

选中工程中使用的 ImageTarget，在其对应的 Inspector 窗口中找到 HiAR Target (Script)，在 Data Set 选项中选择 mydata，然后在 Image Target 选项中选择需要的识别图片。



4. 调整 ImageTarget 的尺寸

ImageTarget 在场景中的尺寸时常需要调整，所以我们给 ImageTarget 添加了 Width 和 Height 两个属性，以方便这一操作。



注意：

由于 Unity 自身导入功能的设计问题，当同一图集重复导入两次时，只有第一次导入内容有效。如果发现第一次导入存在问题，请先删除第一次导入的内容后再导入正确的识别库。

10.7 开发云识别应用

为了解决识别图片的数量限制，以及上线应用不能动态修改识别图片和 AR 内容的问题，我们在 HiAR SDK for Unity 新版本（v1.1.x 及后续版本）中集成了云识别功能。本节主要向大家介绍如何通过 HiAR SDK for Unity 来制作支持云识别的 AR 应用。

下面将介绍如何在 Windows 系统下，使用 HiAR SDK 创建一个简单的支持云识别的 AR 应用。

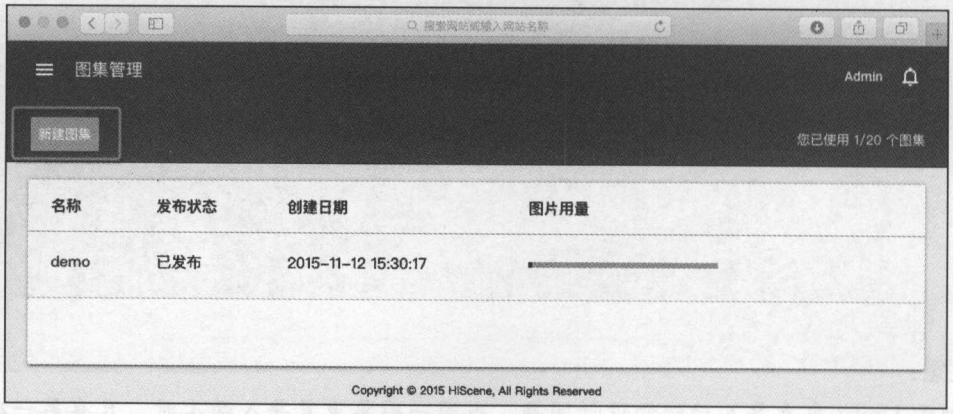
1. 创建图集并关联应用

图集是 HiAR 管理平台提供的一种识别图片管理方式，通过设置图集与应用关联，可以方便地管理应用的云识别内容。

（1）进入管理后台首页，在“图集与识别图片”模块处点击“查看所有图集”，进入图集管理界面。

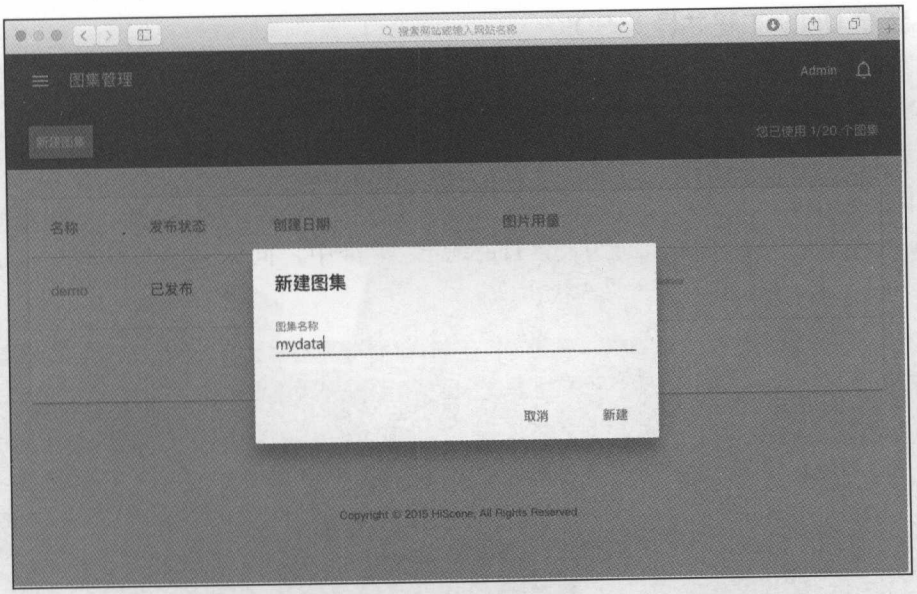


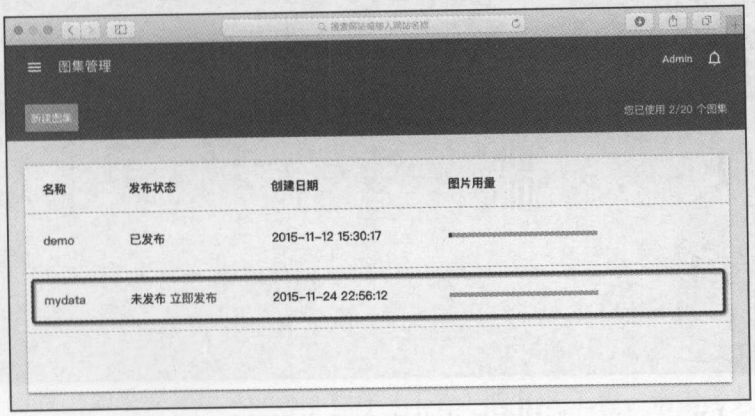
在图集管理界面中，点击“新建图集”按钮。



在弹出的“新建图集”对话框中填写图集名称，点击“新建”完成图集创建。

现在，可以在图集列表中看到刚刚新建的图集。





（2）关联应用，顾名思义，就是将图集关联到应用，只有通过关联操作，在应用进行云识别的时候，该图集中的图片才会被识别。

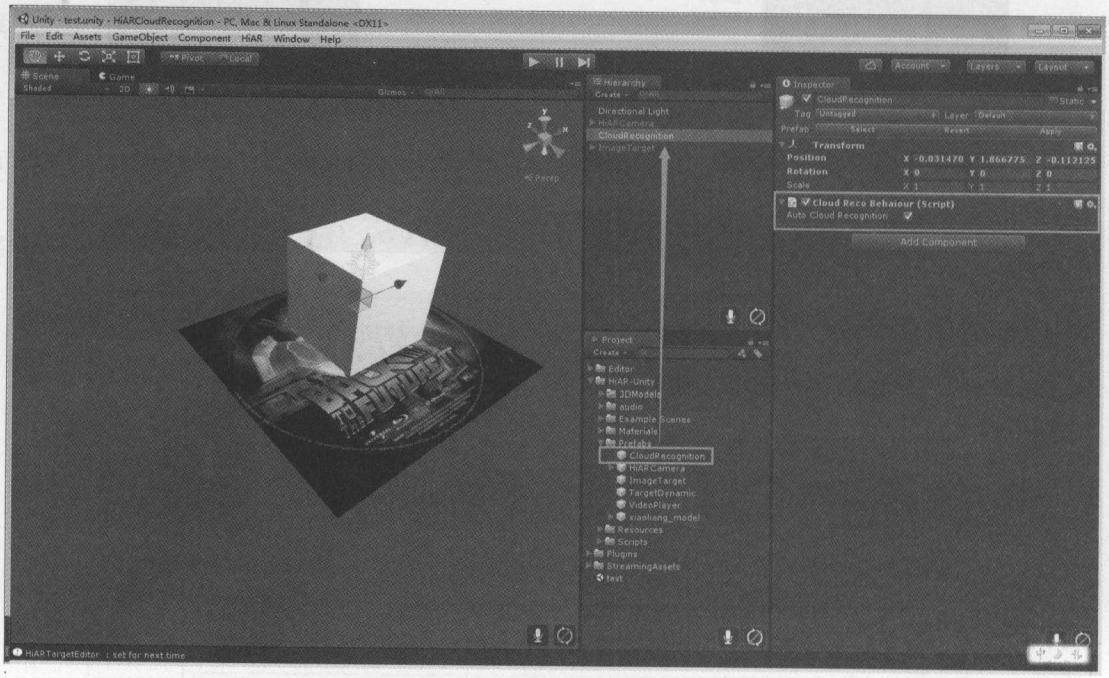


2. 新建 Unity 工程并导入 SDK

运行 Unity 程序，新建一个 Unity 工程，然后将 SDK 包中 lib 目录下的 hiar_sdk_unity-x.x.x.unitypackage 文件导入该工程。

3. 创建 HiARCamera

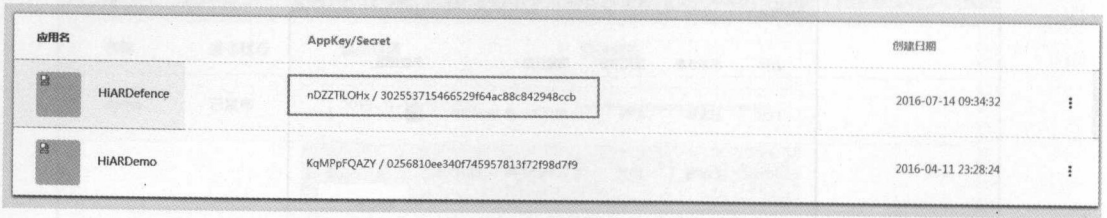
先将工程中默认创建的 Main Camera 删除，再将 Assets/HiAR-Unity/Prefabs 目录下的 HiARCamera 和 CloudRecognition 拖至 Hierarchy 窗口中，同时确保 CloudRecognition 的“Auto Cloud Recognition”处于勾选状态。

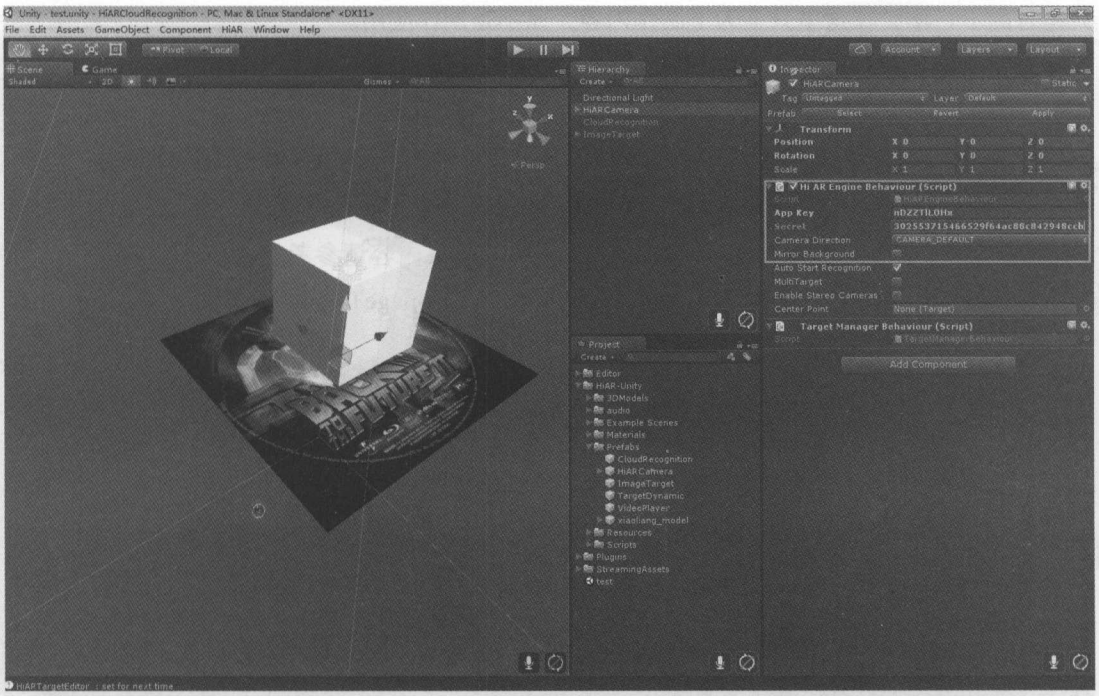


“Auto Cloud Recognition”处于选中状态表示应用运行时会自动进行云识别，否则不进行云识别。

4. 设置 AppKey 和 Secret

在 Hierarchy 窗口中选中 HiARCamera，在其对应的 Inspector 窗口中找到 AppKey 和 Secret 选项，并填写信息。





注意：

如果没有指定正确的 AppKey 和 Secret，应用可能无法正常运行，也不能进行云识别，因为云识别需通过该 AppKey 和 Secret 与你在 HiAR 管理后台创建的应用对应，以此来确定识别图集。

到这一步，应用的功能都已经实现完成，与本地识别应用的区别在于不需要添加本地识别库以及 AR 内容，如果要导出 Android 程序（工程）或 iOS 工程，请参考其他章节。

10.8 制作云识别内容

本节将介绍如何制作并上传 AR 内容到 HiAR 管理后台，以达到云识别并展示 AR 内容的效果。

在开始制作之前，我们先了解一下 HiAR SDK for Unity 云识别的流程及原理。简单说就是先将制作好的“AR 内容”上传到云端，并与入库的“识别图片”相关联，“应用端”通过“云识别服务”识别出对应的“识别图片”后，下载相关联的“AR 内容”并在“应用端”进行展示。

1. 准备工作

（1）首先创建 Unity 工程并导入 SDK，这一步骤与本地识别应用的制作过程完全一样，

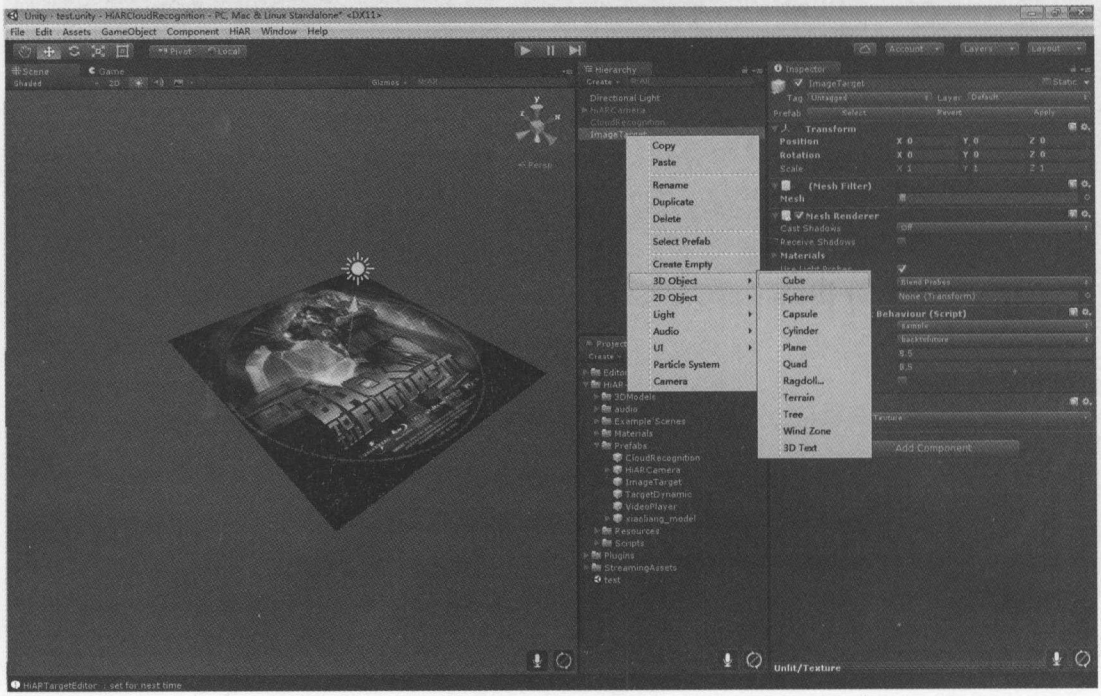
只是目的不一样，在这里只是为了制作内容，而不是制作应用。

（2）将新增的需要识别的图片上传到 HiAR 管理后台，发布成功后下载包含该识别图片的识别包，并将其导入新建的工程，制作对应的 AR 内容。

2. 制作 AR 内容包

AR 内容包是 Unity AssetBundle 的 zip 压缩包，HiAR SDK for Unity 通过云识别后可将该识别包在运行时加载显示。关于 AssetBundle 的概念及内容请参考 Unity 的官方说明。

以云识别一幅图片并显示 Cube 模型为例，先创建 ImageTarge 到 Hierarchy 目录下，并设定好识别图片，在该 ImageTarget 下创建一个 Cube。

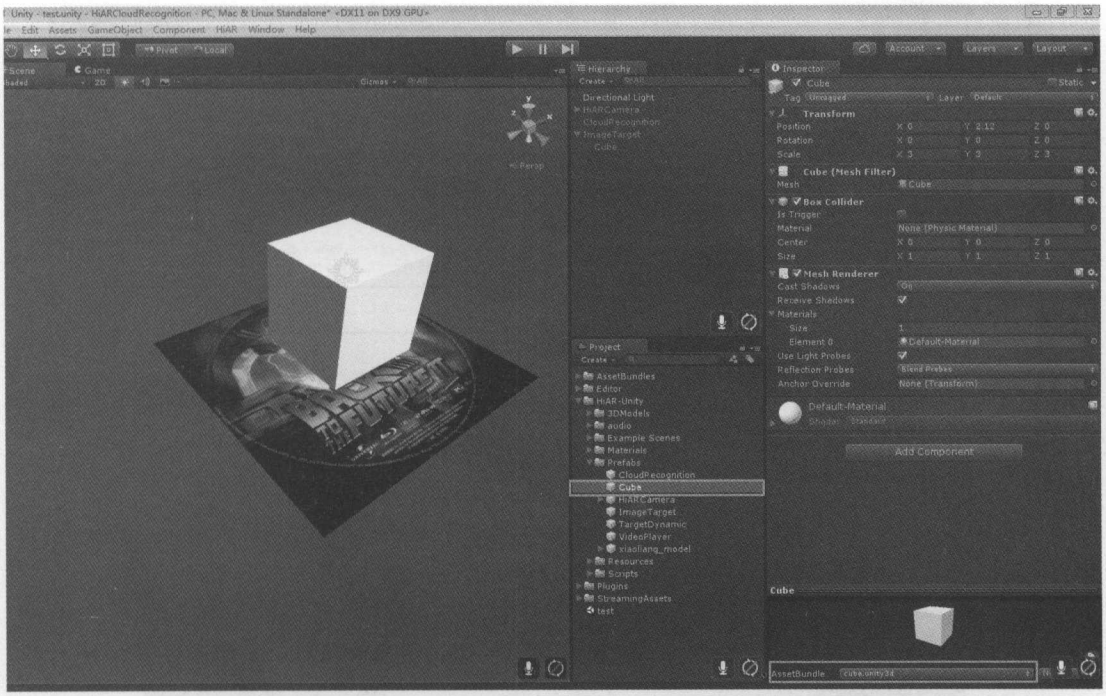


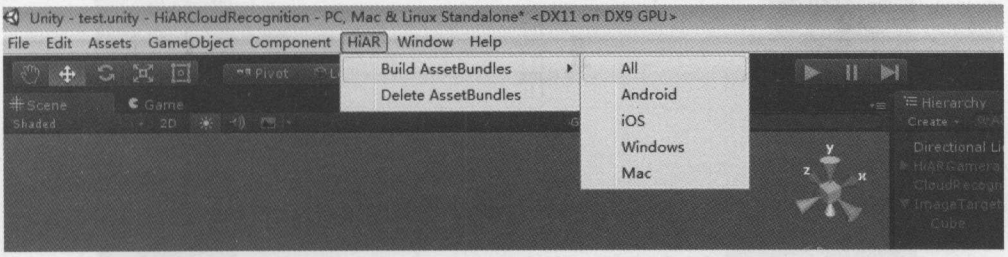
然后，将 Cube 拖到 Prefabs 目录下，选中 Prefabs 中的 Cube 后，在 Inspector 窗口中设置 AssetBundle 名为 cube 及扩展名为 unity3d。此处的 AssetBundle 名可以根据实际情况命名，但扩展名必须为 unity3d。

最后选择菜单 HiAR → Build AssetBundles → All 项，生成 AR 内容包。刷新 Asset 目录后，可以看到在该目录下生成了 AssetBundles 子目录，该子目录下有 Android、iOS、Windows 和 Mac 四个子目录，分别存放不同平台的 AR 内容包。

注意：

Unity 针对不同的平台有不同的 AssetBundle 文件格式，所以如果要支持多平台，必须为每个平台都生成对应的 AssetBundle。





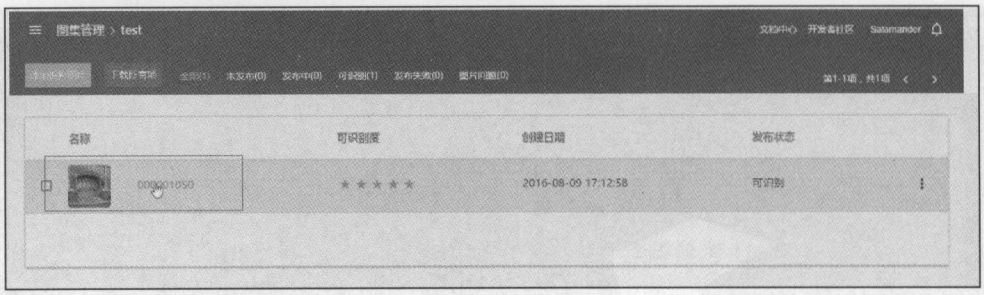
注意：

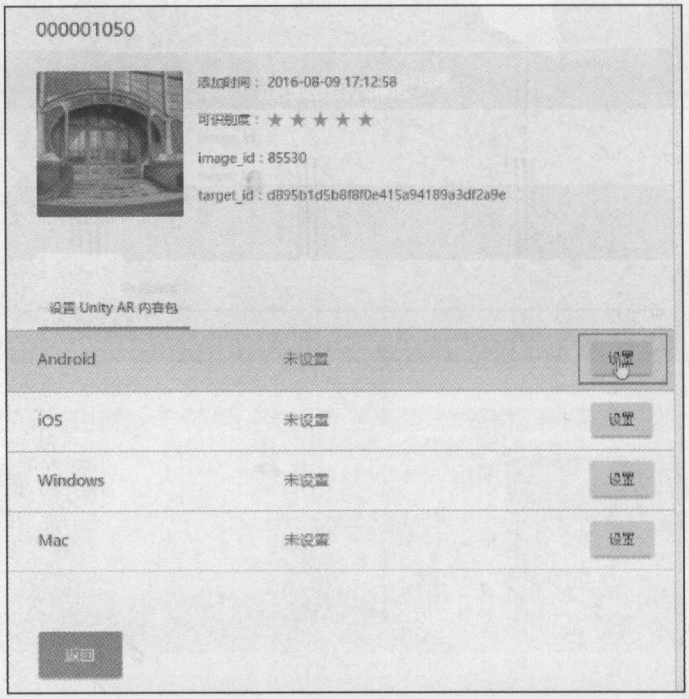
从 1.2.0 版本开始，只需要把要显示的模型做成 assetbundle 并上传就可以了。

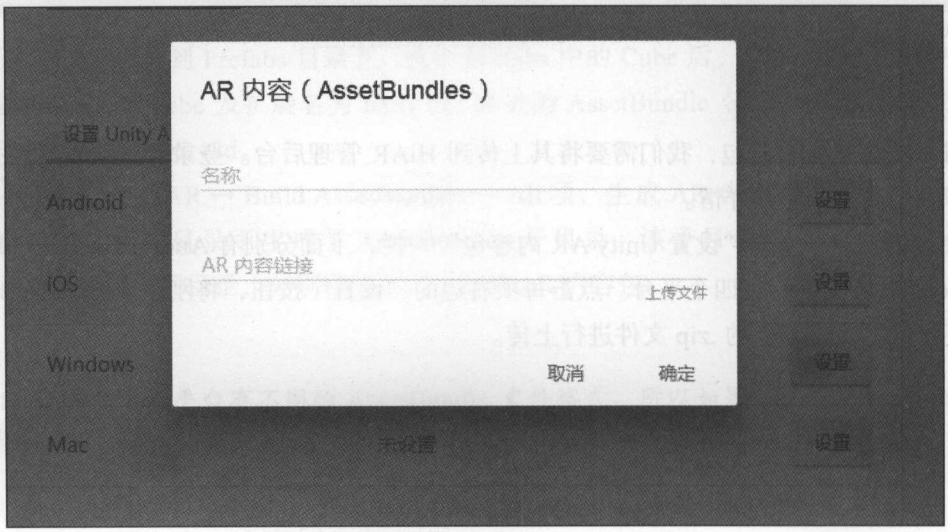
3. 上传 AR 内容包

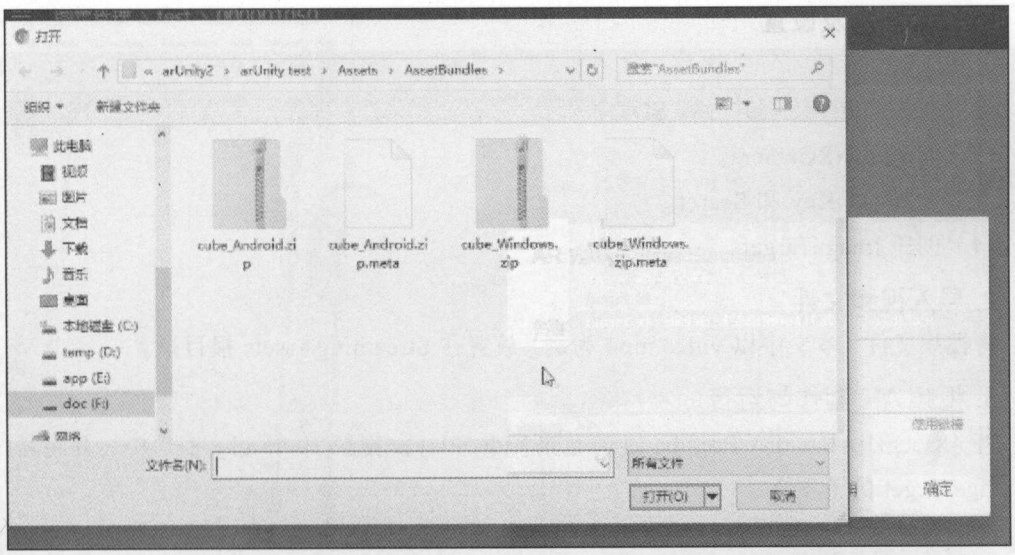
准备好了 AR 内容包，我们需要将其上传到 HiAR 管理后台。登录管理后台后，找到对应的识别图片，查看其详情。

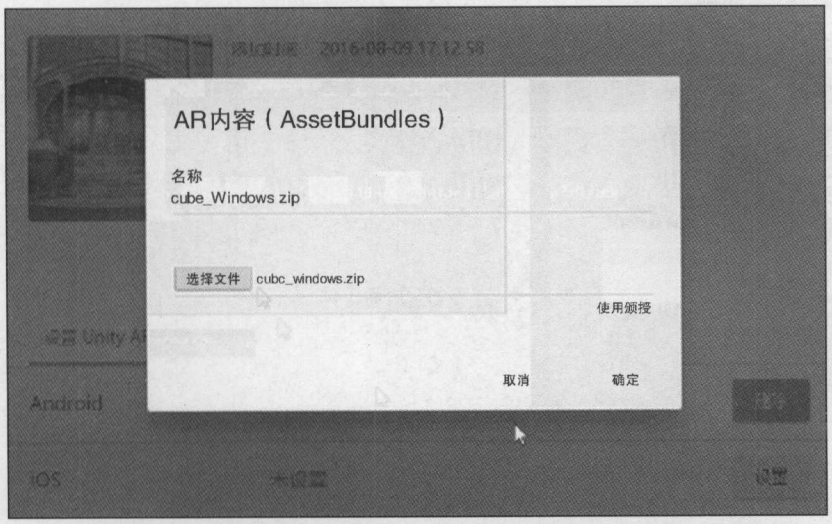
在详情页的下方有“设置 Unity AR 内容包”一栏，下面分别有 Android、iOS、Windows 和 Mac 四项，分别代表四个平台。点击每项右边的“设置”按钮，将刚生成的 AssetBundles 目录下对应平台目录下的 .zip 文件进行上传。


图集管理 > test
名称
可识别度
创建日期
发布状态
000001050
★ ★ ★ ★ ★
2016-08-09 17:12:58
可识别


000001050
添加时间：2016-08-09 17:12:58
可识别度：★ ★ ★ ★ ★
image_id：85530
target_id：d895b1d5b8f8f0e415a94189a3df2a9e
设置 Unity AR 内容包
Android 未设置 设置
iOS 未设置 设置
Windows 未设置 设置
Mac 未设置 设置
返回


AR 内容（AssetBundles）
名称
AR 内容链接
上传文件
取消
确定
Android
iOS
Windows
Mac
设置





制作和上传 AR 内容包可以根据实际应用情况只选择需要的平台，并不是每次都需要在所有平台上生成或上传。

4. 应用自测

为了确保新上线的内容可以正常工作，可以通过已上线的对应应用直接识别图片以测试效果。

10.9 创建视频 AR 应用

可以将视频作为 AR 内容，叠加到图片上并进行播放。

1. 基础开发及设置

先完成以下工作：

（1）新建 Unity 工程并导入 SDK。

（2）创建 HiARCamera。

（3）设置 AppKey 和 Secret。

（4）创建 ImageTarget。

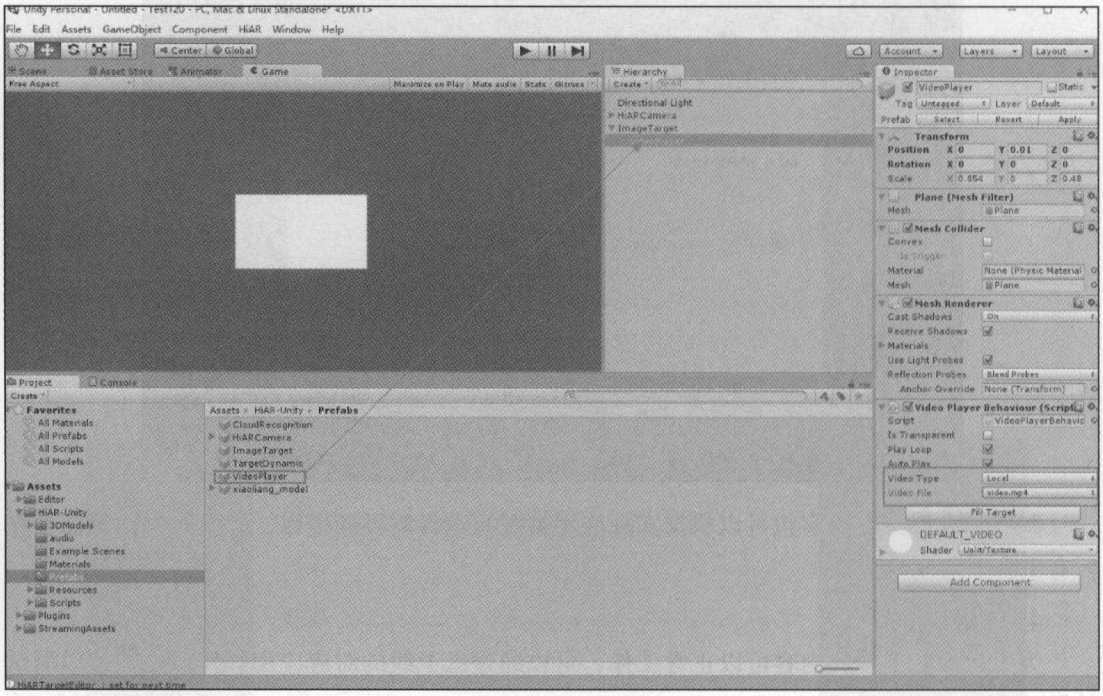
2. 导入视频文件

将视频文件（本文中以 video.mp4 为例）放置在 StreamingAssets 根目录。

3. 视频 AR 制作与运行

将 Assets/HiAR-Unity/Prefabs 目录下的 VideoPlayer 拖至 Hierarchy 窗口中，并将其设置为 ImageTarget 的子项。

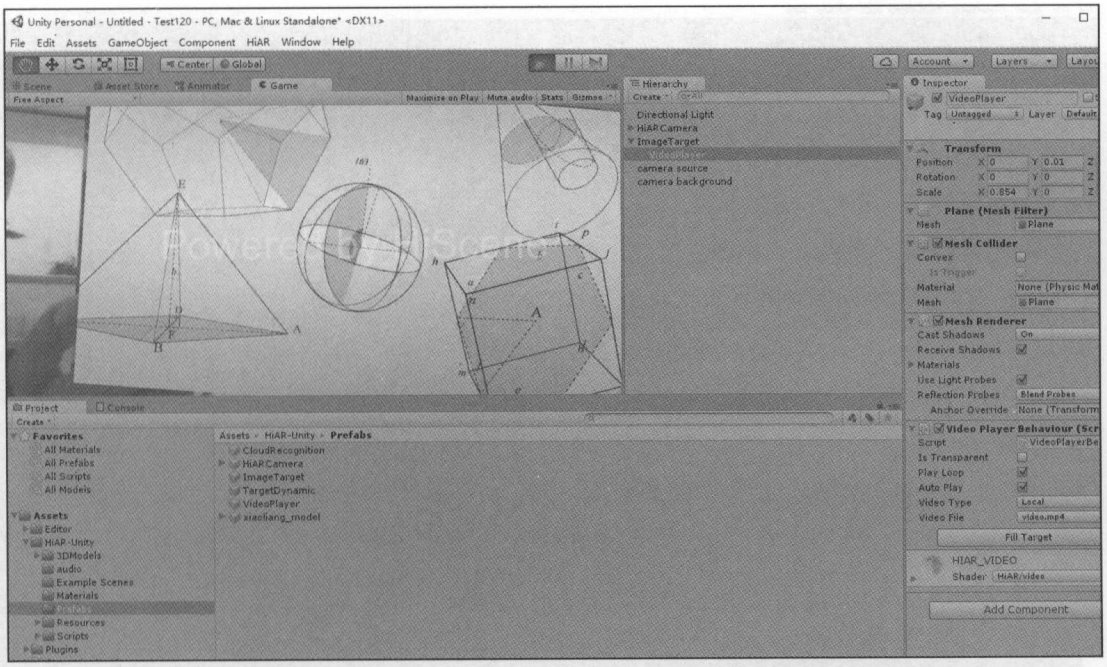
在 Hierarchy 窗口中选中 VideoPlayer，在其对应的 Inspector 窗口中找到 Video Player Mono Behaviour。然后在 Video File 选项中选择 video.mp4。



4. 调试运行

上述操作完成后，需要在 Unity 环境下运行以查看效果。在运行之前，请确保你的电脑已安装了摄像头。

点击运行按钮，摄像头将启动并采集画面；将摄像头对准 apple 图片，识别成功后将在图片上叠加视频，播放的内容为 video.mp4。



10.10　使用 HiAR 创建涂色类 AR 应用

本节将介绍如何使用 HiAR SDK 创建一个简单的涂色类 AR 应用。

1. 基础开发及设置

先完成以下工作：

（1）新建 Unity 工程并导入 SDK。

（2）创建 HiARCamera。

（3）设置 AppKey 和 Secret。

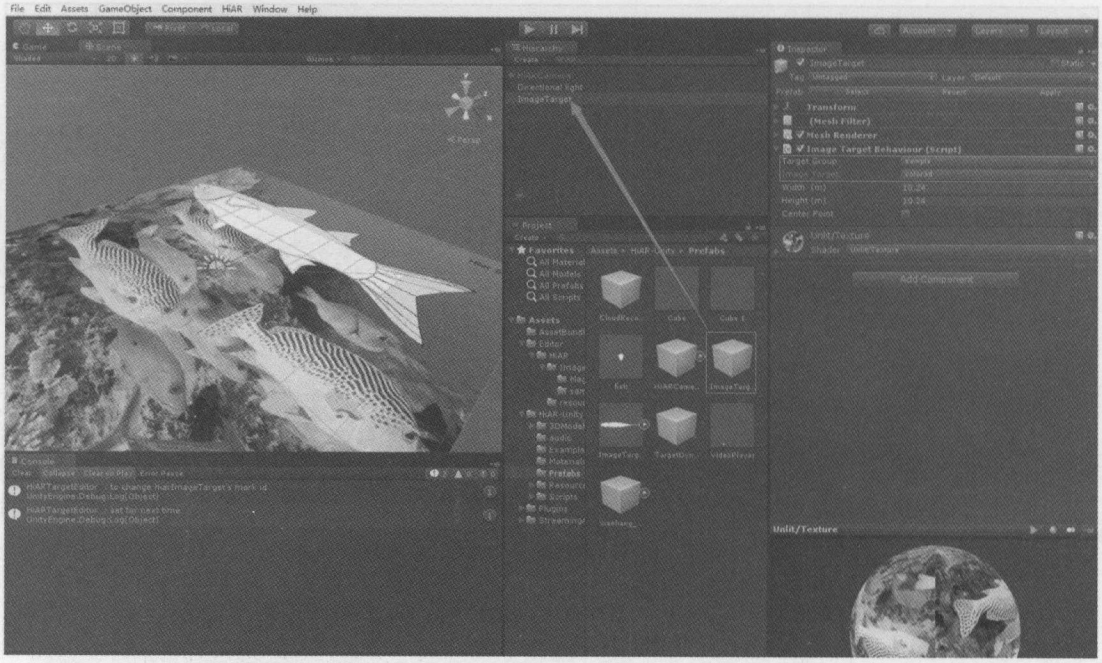
（4）创建 ImageTarget。

2. 设置 ImageTarget

在 Hierarchy 窗口中选中 ImageTarget，在其对应的 Inspector 窗口中找到 HiAR Target Mono Behaviour（Script），在 Data Set 选项中选择 color3D。此时，编辑窗口中将显示一张鱼的图片，这是刚刚设置的识别图片。（说明：color3D 是 SDK 中内置的本地识别包，供教程使用。）

3. 设置 3D 模型

本节使用的是一个鱼的 3D 模型，你可以在 Assets/HiAR-Unity/3DModels 目录下找到名为 fish 的文件。将 fish 拖至 Hierarchy 窗口中并设置为 ImageTarget 的子项，适当调整位置和大小。

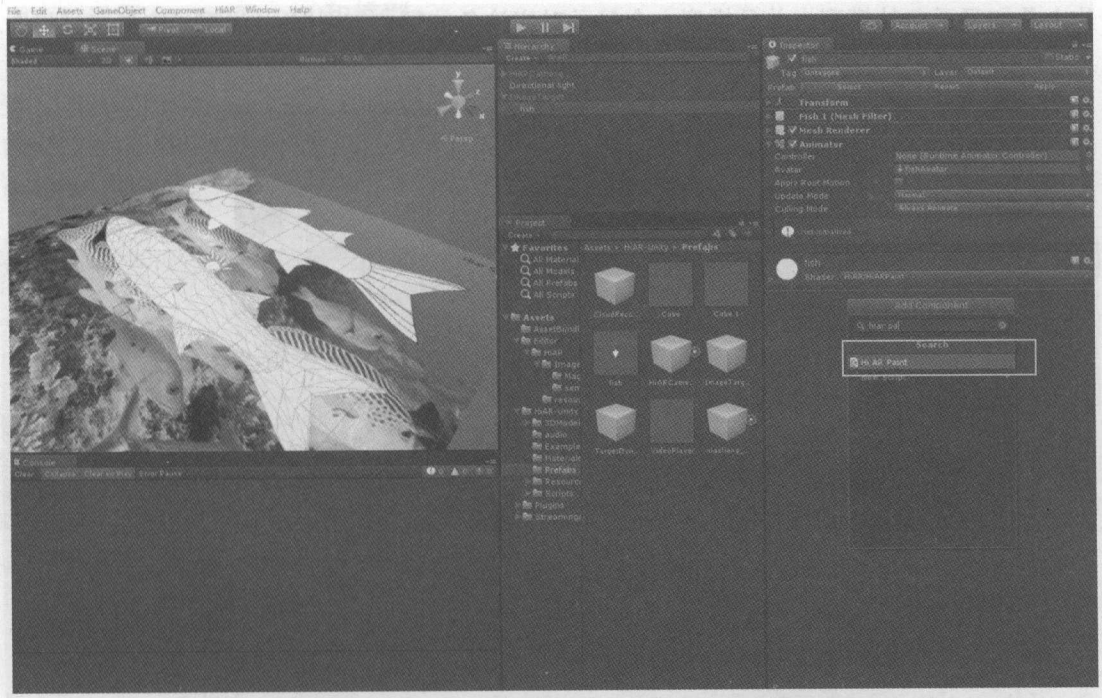


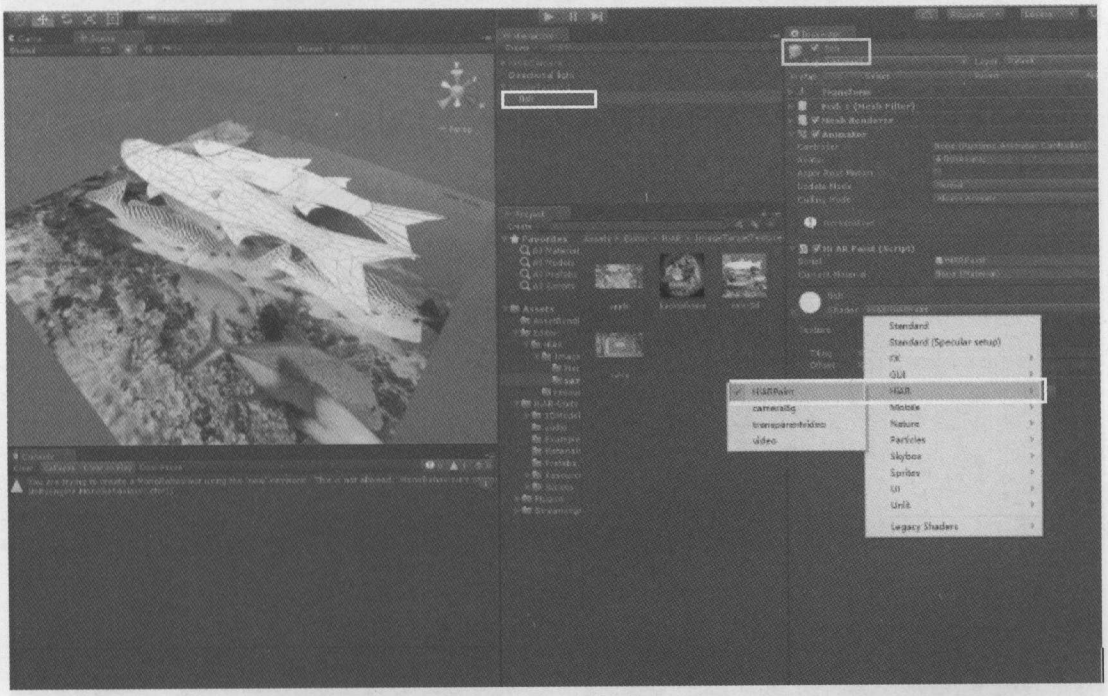


点击 ImageTarget 下的 fish，在其对应的 Inspector 窗口中点击 Add Component，依次选择 Scripts → HiAR Paint。

4. 设置 Shader

在 3D 文件 fish 的同级目录下找到 Material 文件夹并打开，找到对应的材质文件 fish。点击 fish，在对应的 Inspector 窗口中找到 Shader 选项，依次设置为 HiAR → HiARPaint。





5. 调试运行

上述操作完成后，需要在 Unity 环境下运行以查看效果。在运行之前，请确保你的电脑已安装了摄像头。

点击运行按钮，摄像头将启动并采集画面；将摄像头对准鱼的图片，识别成功后将在图

片上叠加显示鱼的模型，并且模型上会显示对应涂色的颜色与图案。（说明：这里已经将图片打印出来，并且在一条鱼的空白处涂上了颜色。）



10.11 多图识别

本节将介绍如何使用 HiAR SDK 的多图识别功能来识别并跟踪多个目标。

1. 基础开发及设置

先完成以下工作：

（1）新建 Unity 工程并导入 SDK（SDK 版本不低于 1.1.12）。

（2）创建 HiARCamera。

（3）设置 AppKey 和 Secret。

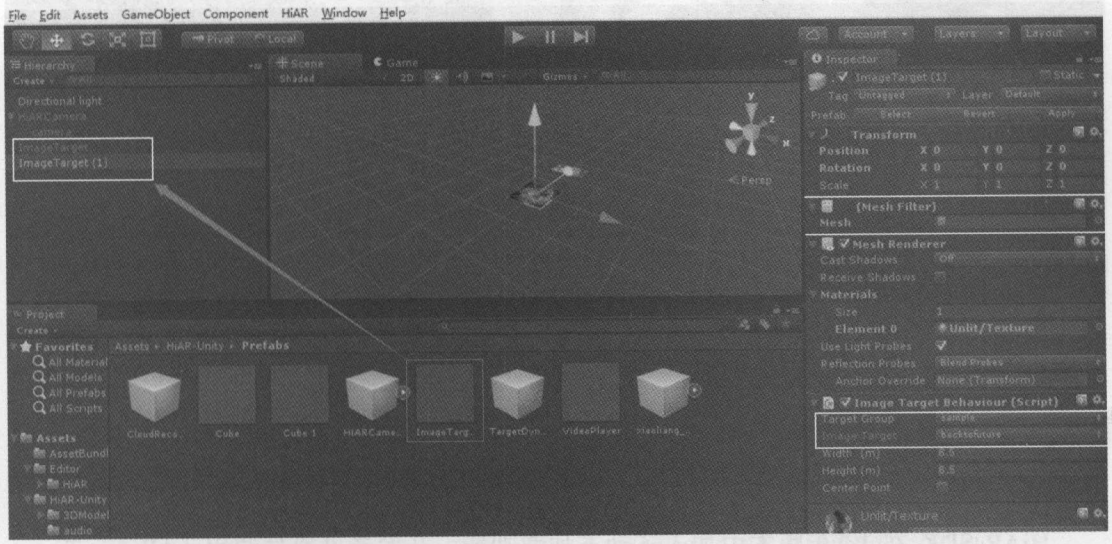
2. 开启多图识别

在 HiARCamera 的 Inspector 面板中找到 MultiTarget 并勾选。



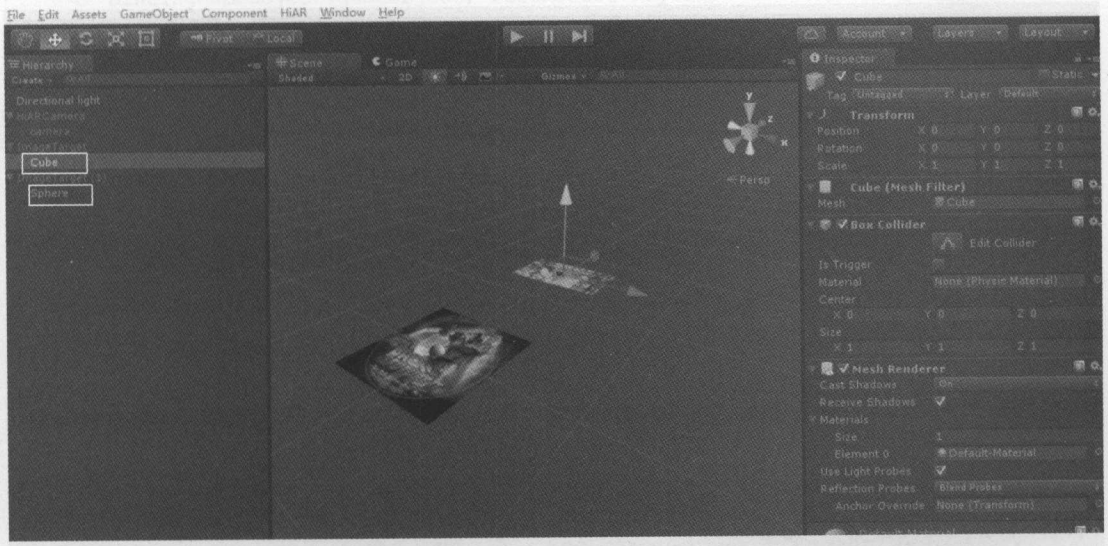
3. 设置 ImageTarget 和模型

将 Assets/HiAR-Unity/Prefabs 目录下的 ImageTarget 拖至 Hierarchy 窗口中，并将其设置为 HiARCamera 的子项。同时添加两个 ImageTarget，在其对应的 Inspector 窗口中找到 HiAR Target Mono Behaviour（Script），在 Data Set 选项中选择 sample，然后在 ImageTarget 中选择识别图片。



4. 设置模型

分别为两个 ImageTarget 设置两个简单的模型。



5. 调试运行

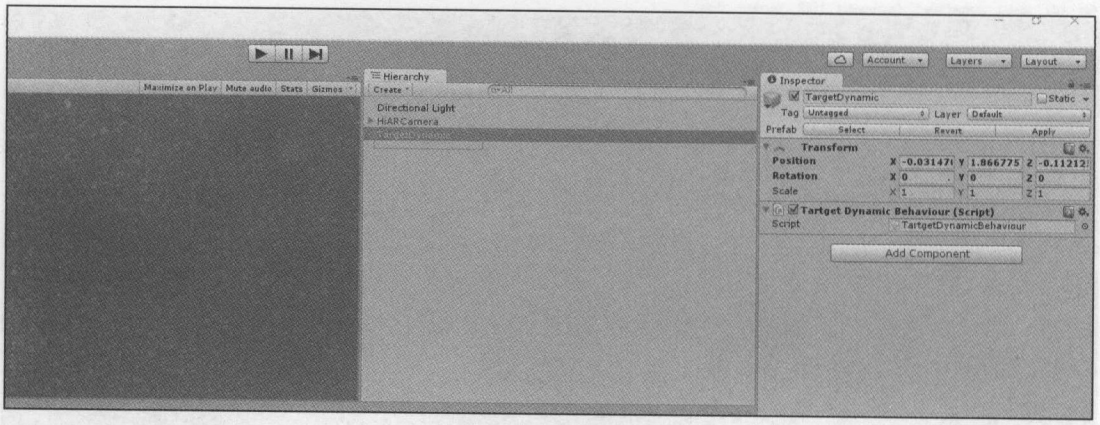
点击运行按钮，摄像头将启动并采集画面；将摄像头对准两张识别图片，识别成功后将同时在两张图片上叠加对应的模型。



10.12 动态加载

HiAR SDK 在 1.2.0 版本中加入了动态加载功能，开发者可通过调用 AddImageTarget AddImageTargetFromGroup 接口实现动态加载识别目标并进行识别跟踪的功能。

1. 将 TargetDynamic Prefab 拖至 Hierarchy 窗口中



2. 加载识别文件

例如，可以通过“点击事件”加载 sample 中的 apple 图片。

```
void OnGUI()
{
    if(GUI.Button(new Rect(30,30,200,80),"Add Key"))
    {
      AddImageTarget(System.IO.Path.Combine(Application.streamingAssets Path,
```

```
"HiAR/sample"), "e58931b38b636696f65caafb0ed80f0b",854,480);
        }
    }
```

3. 设置内容

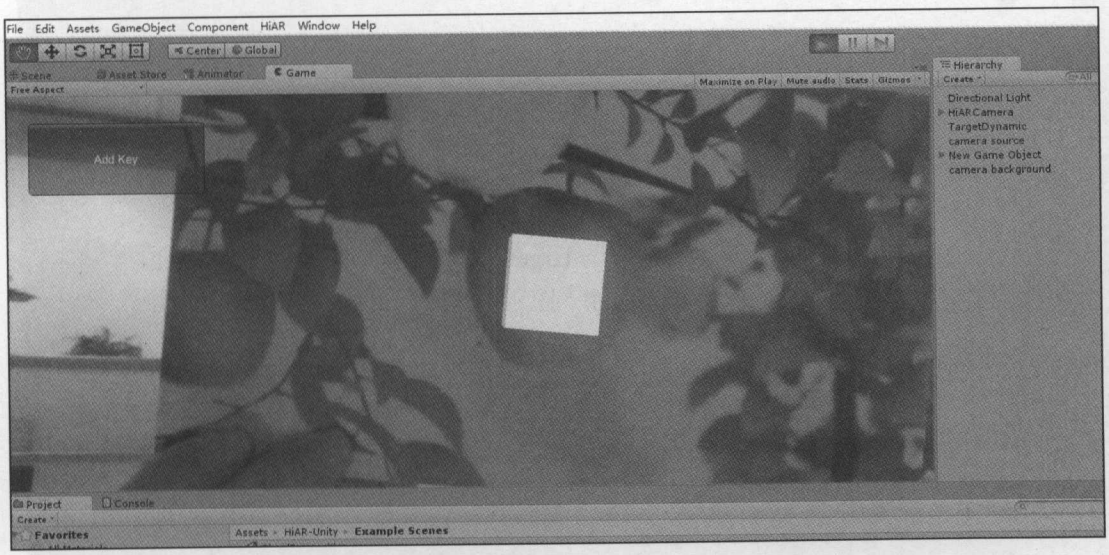
当识别到动态添加的 key 后，可在 OnDynamicReco 中设置相应展示内容。例如识别后设置显示一个 cube。

```
public override void OnDynamicReco(RecoResult recoResult)
{
    GameObject gameObject = null;
    gameObject = new GameObject();
    if (recoResult.keyType == KeyType.IMAGE)
    {
        gameObject.AddComponent();
    }

    Target target = gameObject.GetComponent();
    target.PixelWidth = recoResult.Width * 0.01f;
    target.PixelHeight = recoResult.Height * 0.01f;
    gameObject.transform.parent = transform.parent;
    gameObject.SetActive(true);

    if (recoResult.KeyId.Equals("e58931b38b636696f65caafb0ed80f0b"))
    {
        GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
        cube.transform.parent = gameObject.transform;
    cube.transform.localScale = new Vector3(1.0f, 1.0f, 1.0f);
    }
    bindingGameObject(gameObject, recoResult.KeyId);
}
```

4. 运行以查看效果



10.13 设置场景中心点

1. 选择一个 target 为中心点

在 Hierarchy 窗口中选择 ImageTarget，在其对应的 Inspector 窗口中找到 Center Point 选项，勾选则表示选择了这个 ImageTarget 为中心点。



注意：

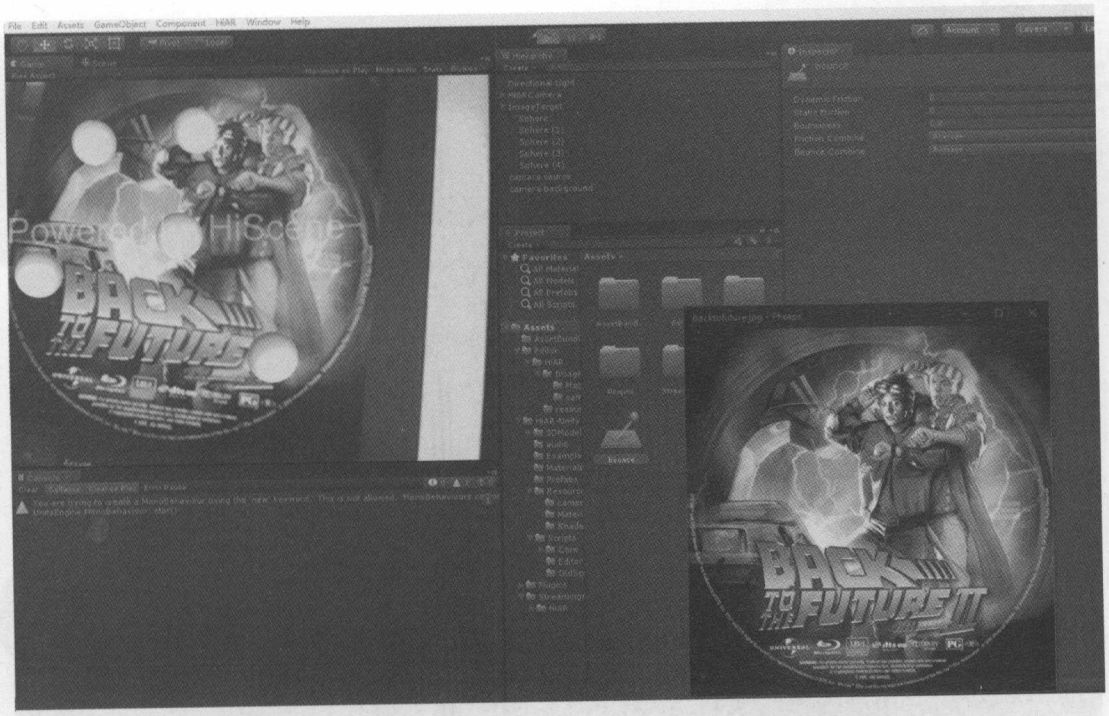
将 ImageTarget 拖到 HiARCamera 的 CenterPoint 上，效果与在 target 上勾选 Center Point 一样。

2. 将 target 固定到一个位置，使重力方向始终向下

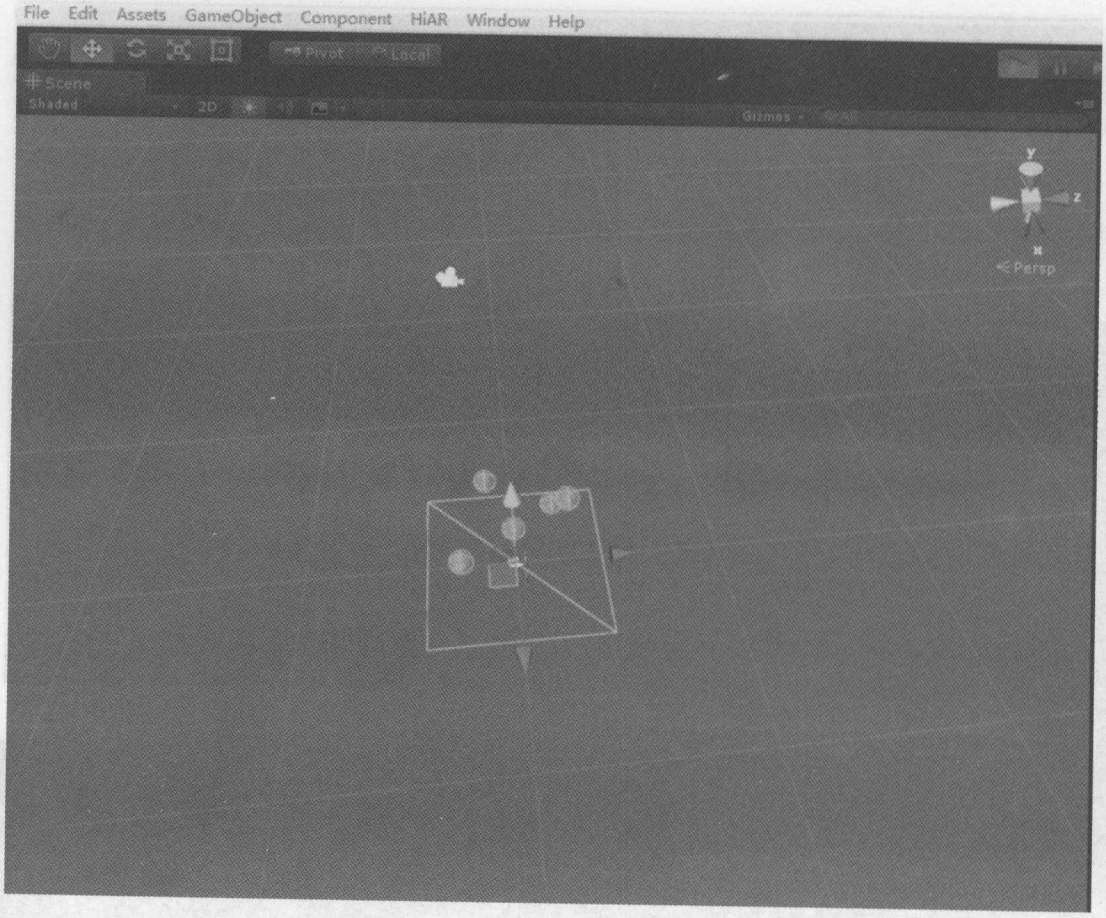
有些时候我们需要呈现一些物理碰撞的效果，例如：

但不设置中心点的时候，重力方向不与 target 垂直（target 会不断变化），如下图所示。

而设置该 target 为中心的时候，效果如下图所示。


File Edit Assets GameObject Component HiAR Window Help
Powered by HiScene
BACK TO THE FUTURE
BACK TO THE FUTURE II


File Edit Assets GameObject Component HiAR Window Help
Pivot
Local
Scene
Shaded
2D
Gizmos
Persp


File Edit Assets GameObject Component HiAR Window Help
Pivot
Local
Scene
Shaded
2D
Gizmos
Persp

第11章 其他 AR SDK 简介

11.1 基于 Wikitude SDK 开发 AR 应用

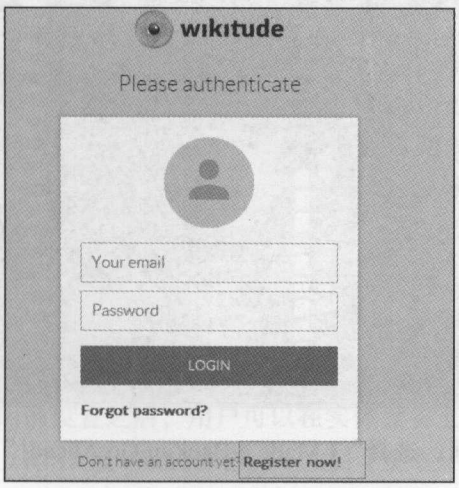
Wikitude 有限责任公司是移动增强现实技术的先驱，其针对智能手机、平板电脑以及智能眼镜所开发的 AR 解决方案获得了多项大奖。

Wikitude 完全自主研发的 AR 技术是以核心产品 Wikitude AR SDK 以及一系列辅助工具的方式来推向市场的，帮助成千上万的商标持有人、广告公司、开发人员以及 AR 技术爱好者获取理想的项目目标。Wikitude 是全球领先的 AR 技术供应商，拥有数以万计的开发者账户以及 AR 应用。

1. 准备 Wikitude 开发环境

（1）打开 Wikitude 网站注册页面 https://targetmanager-cn.wikitude.com。

（2）点击“Register now”，如果已经注册为 Wikitude 用户，则输入邮箱和密码登录 Wikitude 管理后台；如果不是 Wikitude 用户，则点击注册按钮填写注册信息。

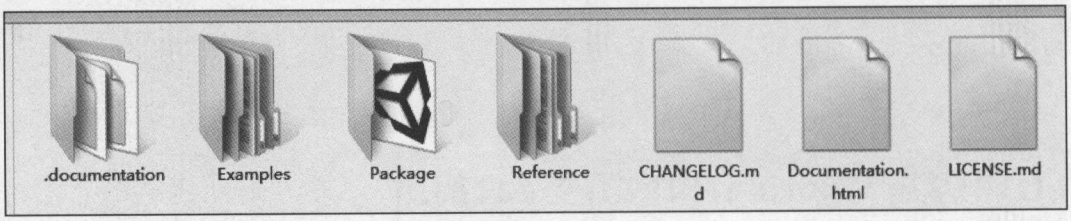


（3）注册完成后会发送初始密码以及免费 Wikitude SDK 试用许可证密钥至用户邮箱，请登录邮箱查看。

（4）填写注册邮箱和密码，登录 Wikitude 目标管理器。

（5）登录成功后可以在右上角的设置中切换语言为中文，这对于国内开发者来说使用起来更为方便。

（6）下载 Wikitude SDK for Unity。访问 http://www.wikitude.com/download/ 进入 Wikitude SDK 下载界面。选择 Wikitude SDK for Unity，点击 Download 默认下载最新版本的 SDK。也可以点击 Release Notes 查看历史版本并选择合适版本进行下载。本节使用最新的 Unity 插件 1.2.1-1.1.0。

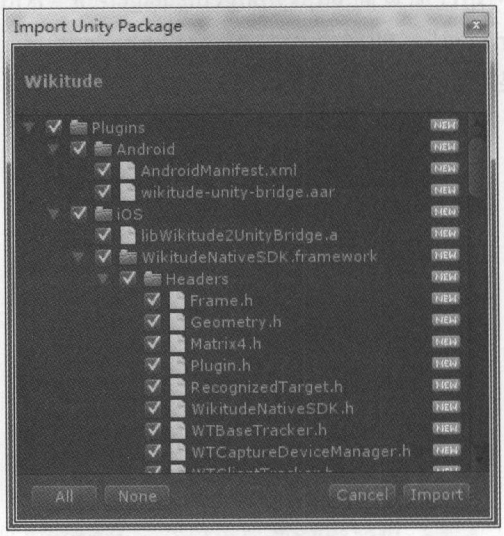


- .documentation 目录：包含使用说明文档、证书等。
- Examples 目录：包含 WikitudeUnityExample。
- Package 目录：包含 Wikitude.unitypackage。
- Reference 目录：包含参考文献。

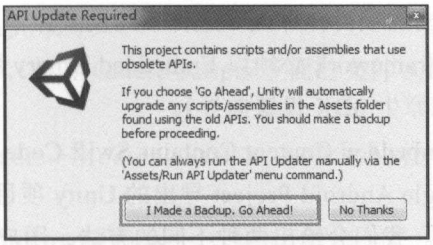
本节使用的 Unity 版本为 5.3.1f1。

2. 导入 Wikitude SDK for Unity 插件

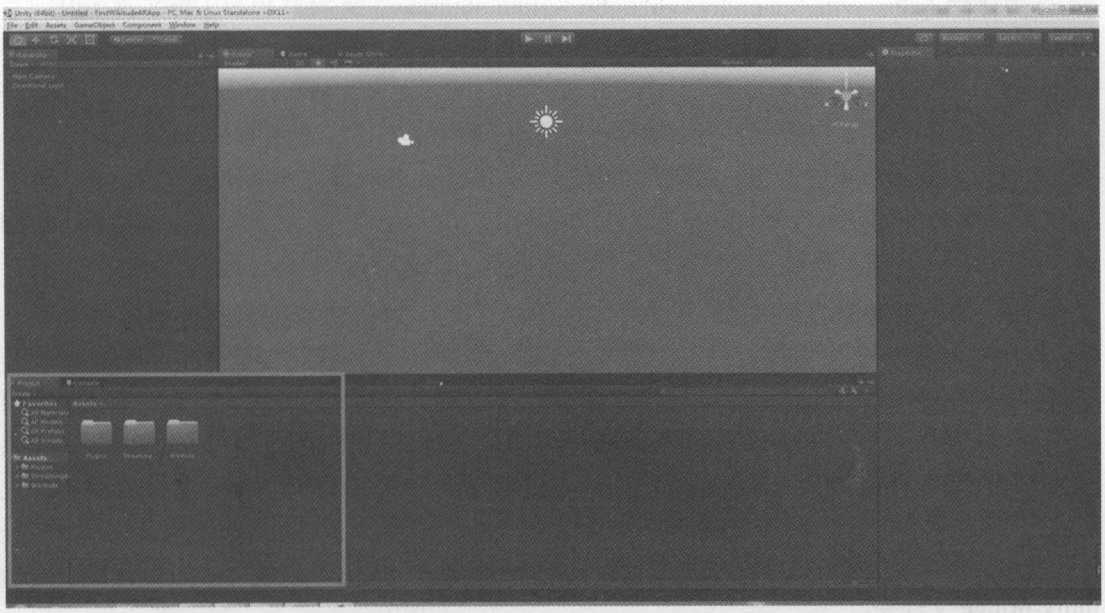
双击 Wikitude.unitypackage 文件，Unity 将加载并显示插件内容，点击 Import 开始导入。也可以在 Unity 中点击 Assets → Import Package → Custom Package... 来导入插件。



如果出现以下提示界面，选择”I Made a Backup.Go Ahead!”按钮即可。



导入完成后，可以在 Project 窗口中看到以下内容。



至此，SDK 导入工作已经完成，可以开始创建你的第一个 Wikitude 工程了。

3. 完成 Unity 插件设置

用户仅需完成几个必要的步骤，就能将 Wikitude Unity 插件添加到既有的 Unity 项目中。本节会详细解释每一步，它们是：

- 将 Wikitude.unitypackage 导入项目。
- 将 WikitudeCamera 和 ClientTracker 或 CloudTracker prefab 添加到 Unity 场景层级中。
- 导出项目，修改 Xcode Build Settings 来完全整合 Wikitude 原生 SDK。

（1）首先导入 Wikitude.unitypackage，Wikitude Unity 插件是标准的 .unitypackage，可通过正常 Unity 包导入流程进行导入。在资源检视面板中，右键点击并选择 Import Package → Custom Package。从弹出的文件浏览窗口中选择下载好的 Wikitude.unitypackage 并点击 Open。这时，所有与 Wikitude.unitypackage 相关的内容都会被导入。

（2）导入 Wikitude.unitypackage 之后，其所有组件和脚本都可用于定义个性化 AR 体验。范例部分会更详细地描述这些 prefab 和脚本的使用方法。

（3）在 Unity 完成应用的设置之后，用户可以在实体设备上对其进行测试。这时，用户需要将其作为 Xcode 项目（iOS）或 Android Studio 项目（安卓）导出。

iOS 开发者需要打开导出的项目，并人工进行下列设置：

- WikitudeNativeSDK framework 必须以 Embedded Binary 形式添加（欲了解更多信息，请参考 Wikitude iOS 原生 SDK 设置向导）。
- 必须将 Build 选项 Embedded Content Contains Swift Code 设置为 YES。

不过很遗憾，作为 Google Android Project 导出的 Unity 项目无法立即使用。用户需要进行人工设定，具体方法如下。我们会给出两种不同的方法，用户选择其一即可。我们推荐用户尝试第一种方法，但对于某些用户，第二种方法也许更好。

【方法一】

1）在 Unity 中，点击 File | Build Settings ...，勾选 Google Android Project 并点击 Export。

2）打开 Android Studio 并点击 Import Project（Eclipse ADT，Gradle）。

3）在文件选择对话框中，找到导出项目的文件夹，并选择以 app 命名的文件夹（请勿选择文件夹 wikitude-unity-bridge）。

4）在下一步中，选择一个全新空文件夹来导入，并点击 Next 。

5）点击屏幕上的 Finish 完成导入。

6）在 Android Studio 中，打开 Gradle Scripts 下面的文件 settings.gradle，并删除 include ':wikitudeunitybridge'。

7）打开文件 build.gradle（Module: YOUR_APP_NAME），并删除 compile project (':wikitudeunitybridge')。

8）点击 Sync Project with Gradle Files。

9）使用文件管理器打开从 Unity 导入 Android Studio 的那个文件夹，并删除 wikitude-unity-bridge 文件夹。

10）在当前文件夹中，找到包名 /src/main/jniLibs，并从每个子文件夹中删除文件 libnativesdk.so。

11）在 Android Studio 中点击 File | New | New Module ...。

12）选择 Import .JAR/.AAR Package 并点击 Next。

13）转到一开始从 Unity 导出的那个文件夹，并找到以 app 命名的子文件夹。

14）在 libs 下面选择文件 wikitude-unity-bridge.aar，并点击 Finish。

15）打开文件 build.gradle（Module: YOUR_APP_NAME），并在 dependencies 下面添加 compile project (':wikitude-unity-bridge')。

16）这时，项目就可以在 Android Studio 中运行了。

【方法二】

1）在 Unity 中，点击 File | Build Settings ...，勾选 Google Android Project 并点击 Export。

2）打开 Android Studio 并点击 Import Project（Eclipse ADT，Gradle）。

3）在文件选择对话框中，找到导出项目的文件夹，并选择以 app 命名的文件夹（请勿选择文件夹 wikitude-unity-bridge）。

4）在下一步中，选择一个全新空文件夹来导入并点击 Next。

5）点击屏幕上的 Finish 完成导入。

6）在 Android Studio 中，打开 Gradle Scripts 下面的文件 settings.gradle，并删除 include ':wikitudeunitybridge'。

7）打开文件 build.gradle（Module: YOUR_APP_NAME），并删除 compile project (':wikitudeunitybridge')。

8）点击 Sync Project with Gradle Files。

9）转到一开始从 Unity 导出的那个文件夹。

10）将文件 [导出文件夹]/wikitude-unity-bridge/libs/wikitude-native-sdk.jar 和 [导出文件夹]/wikitude-unity-bridge/bin/classes.jar 导入到 [Android Studio 导入文件夹]/YOUR-APP-NAME/libs，并将 classes.jar 更名为 wikitude-unity-bridge.jar。

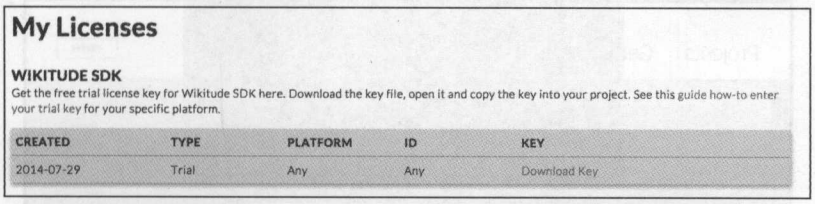
11）回到 Android Studio，打开 build.gradle（Module: YOUR_APP_NAME），并在 dependencies 下面添加 compile files ('libs/wikitude-native-sdk.jar') 和 compile files ('libs/wikitude-unity-bridge.jar')。

12）这时，项目就可以在 Android Studio 中运行了 。

4. 获取免费试用版许可授权

Wikitude SDK 需要有效的授权许可代码才能正常运行，授权许可的缺失会屏蔽 AR 视图，使内容无法正常显示。用户会在屏幕上看到印有 License Key Missing 字样的水印，所有的 JavaScript API 将被忽略而不会执行。

下载 Wikitude SDK 的时候，用户会被带到授权许可生成页面。在此页面，试用版授权许可代码会自动生成，用户也可登录注册邮箱查找，注册 Wikitude 账户时，会给用户发送一封 Wikitude SDK Trial License 的邮件。



将授权许可码复制到 app 中，这会解锁 Wikitude SDK 的试用模式。试用模式包括 Wikitude SDK 的所有功能，但会在屏幕上显示 Trial 水印字样，每个试用授权许可码可用于所有开发平台的所有应用 ID，用户可在多个 app 中使用同一个试用授权许可码。

要在 Wikitude Unity 插件中使用授权许可码，请将授权码粘贴到 WikitudeCamera prefab 的文本区域内。

11.2 基于 VoidAR SDK 开发 AR 应用

太虚 AR SDK（VoidAR）是国内团队自主研发的增强现实引擎。该引擎在 2015 年 9 月

30 日发布了 alpha 0.02 版本，具有“手绘识别、模糊识别、强稳定性、空间构建”的功能特点，并且免费开放使用。

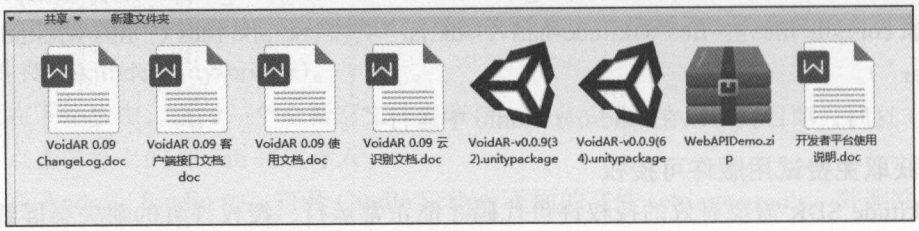
1. 注册成为 VoidAR 用户

（1）在浏览器中打开链接 http://cloudar.ionhammer.cn:8080/develop/html/register.jsp，填写邮箱信息进行注册，注册完成后会跳转到登录页面，使用注册时的邮箱账号和密码即可登录。

（2）若已有太虚账号可在浏览器中打开 http://cloudar.ionhammer.cn:8080/develop/html/login.jsp，即可登录到太虚引擎的云数据库。

2. 下载 VoidAR SDK for Unity

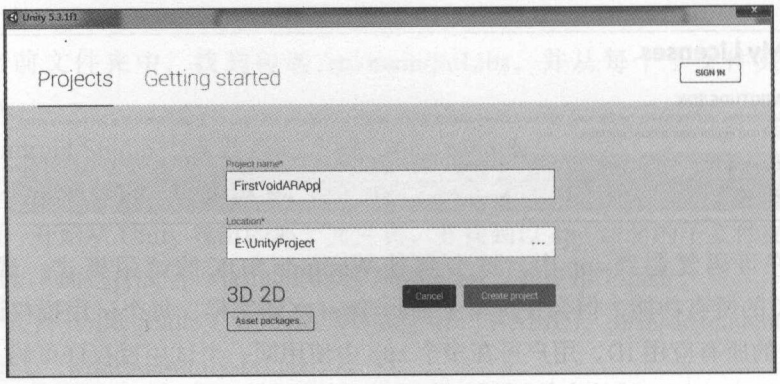
在百度云盘下载界面选择所需版本并下载 VoiAR SDK for Unity，本节下载版本为 v0.09。下载完成后可以获得一个名为 v0.09 的文件夹，打开可以看到如下文件。



可使用 Unity 3D 的 32 位或者 64 位 5.x（建议使用 5.2.0）。

3. 新建 Unity 工程

运行 Unity 程序，并按照步骤新建一个 Unity 工程。当然，也可以直接打开已存在的工程来进行导入工作。

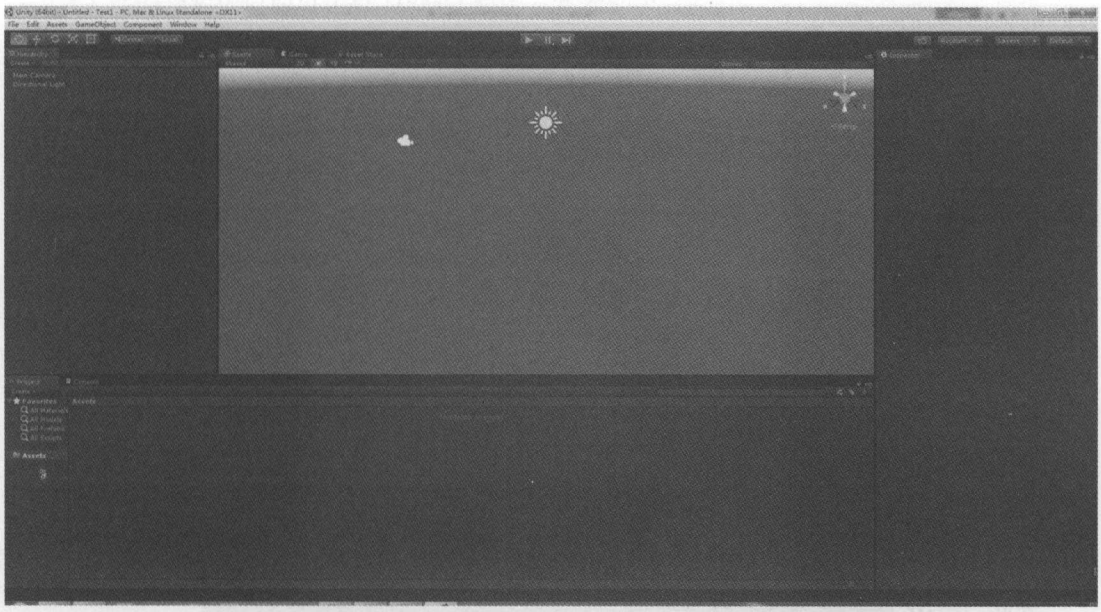


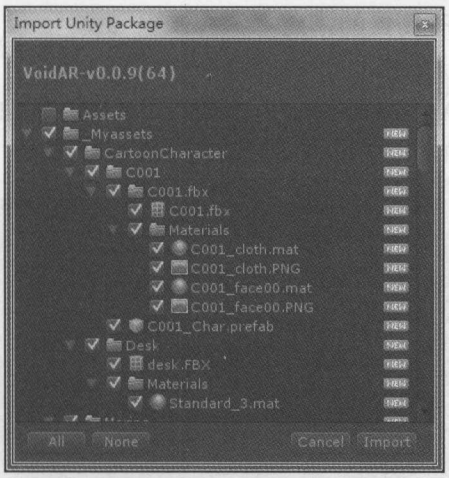
4. 导入 VoidAR SDK for Unity 插件

打开 Unity 3D 之前建立的工程，右键点击 Assets，选择 Import Package，继续选择 Custom Package，选择 VoidAR-v0.0.9(32).unitypackage（对应 32 位的 Unity）或者 VoidAR-v0.09(64).unitypackgae（对应 64 位的 Unity）。

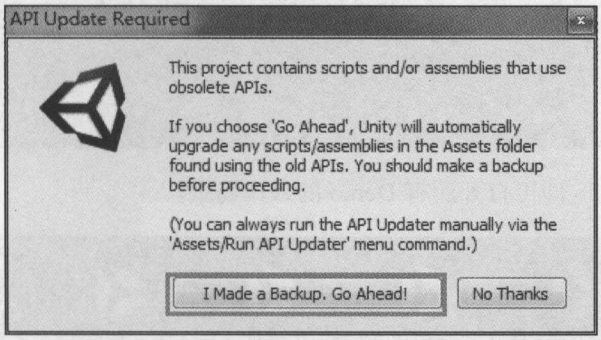
注意：

存放 unitypackage 的路径中不要有中文，否则会导入不成功。

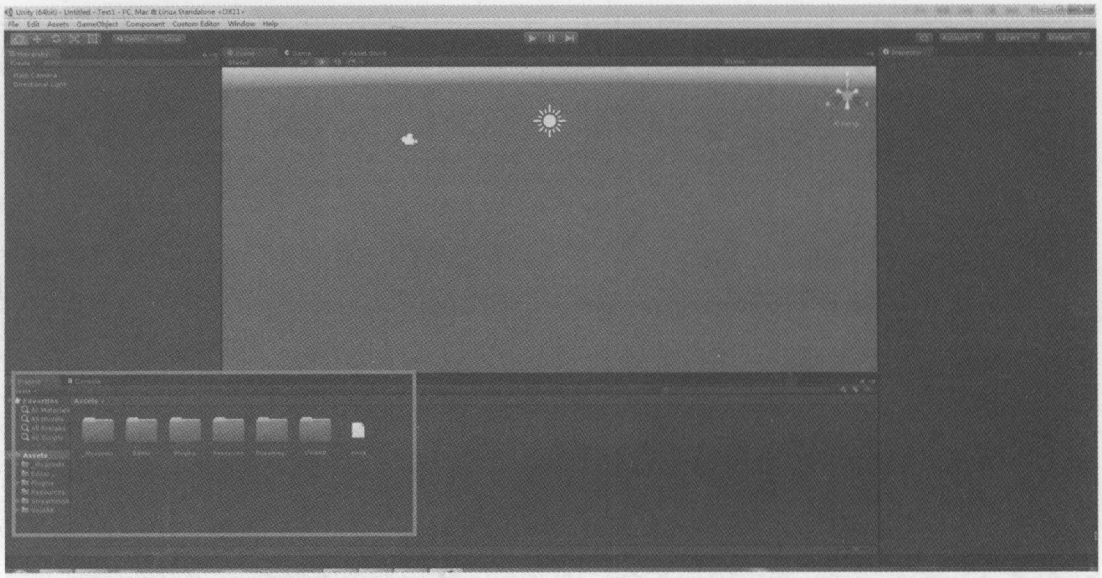




如果出现以下提示界面，点击“I Made a Backup.Go Ahead!”按钮即可。



导入完成后，可以在 Project 窗口中看到以下内容。

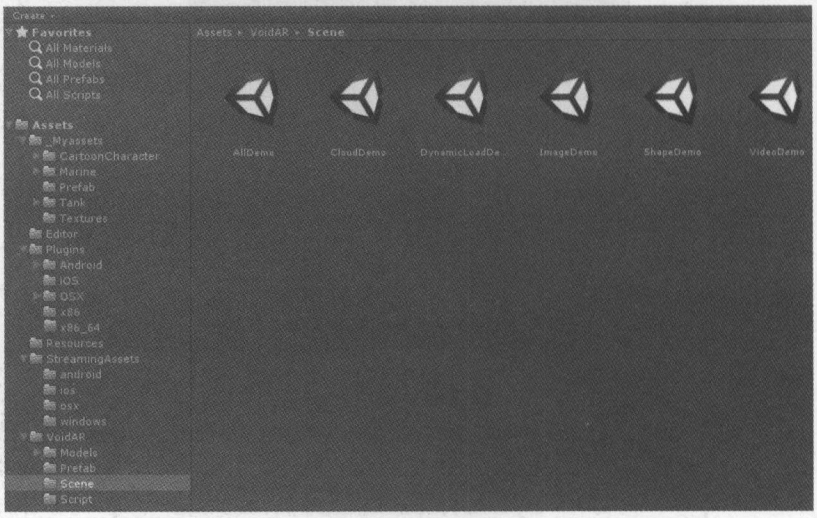


至此，SDK 导入工作已经完成。

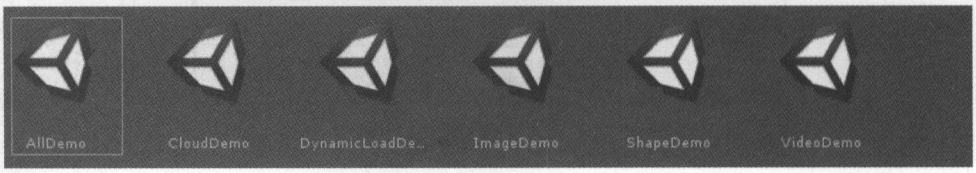
5. 完成第一个 VoidAR 场景的搭建

（1）打开之前创建的 FirstVoidARApp 工程。

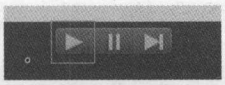
（2）选中 Assets/VoidAr/Scene，文件里存放了 0.09 版本的所有 demo、CloudDemo、DynamicLoadDemo、ImageDemo、ShapeDemo 和 VideoDemo。

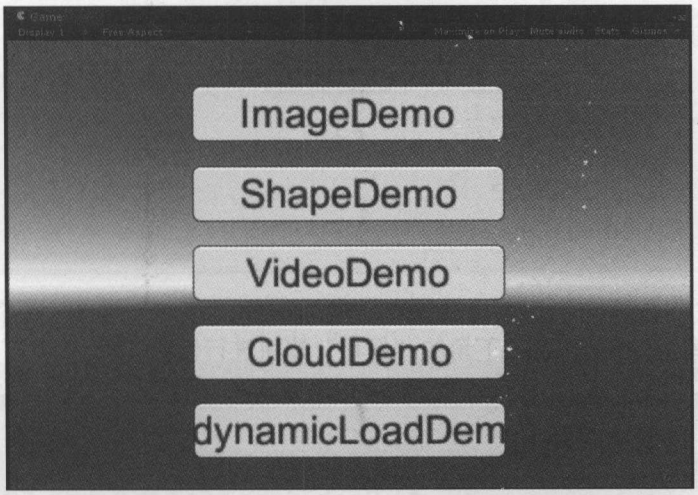


（3）双击 AllDemo 即可打开所有 Demo 的入口场景。



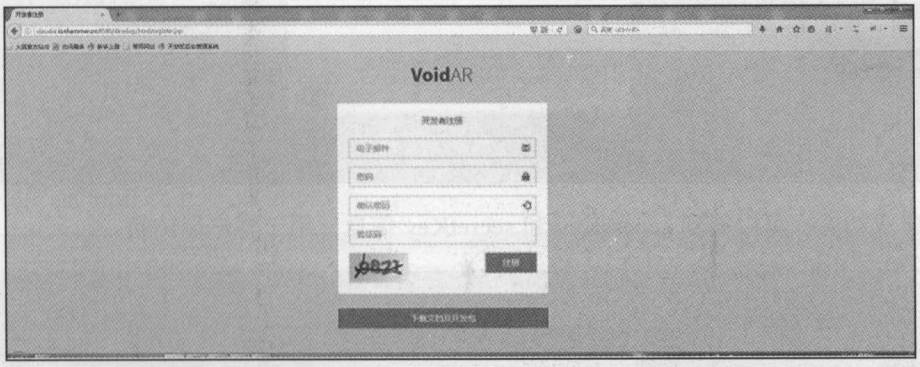
（4）点击开始按钮即可运行 DEMO 程序，效果如下。





6. 开发者平台使用说明

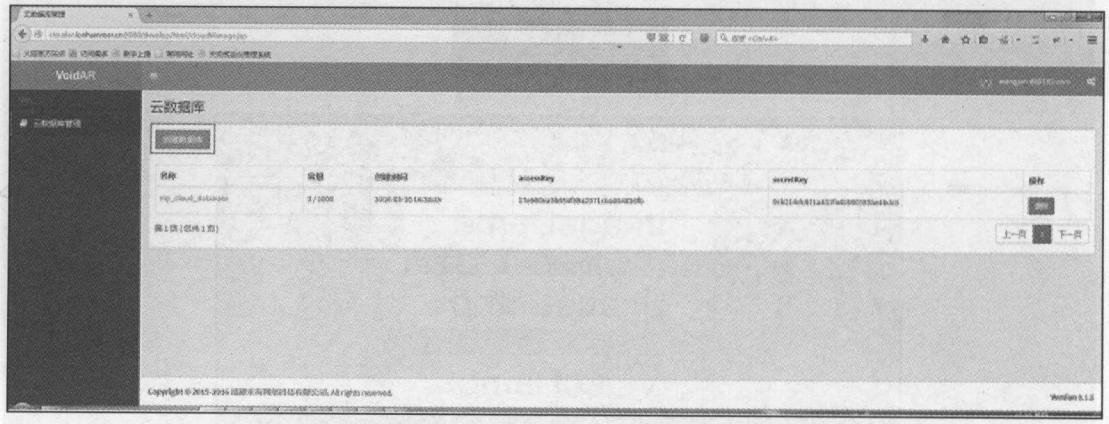
（1）注册。在浏览器中打开 http://cloudar.ionhammer.cn:8080/develop/html/register.jsp 进行注册。



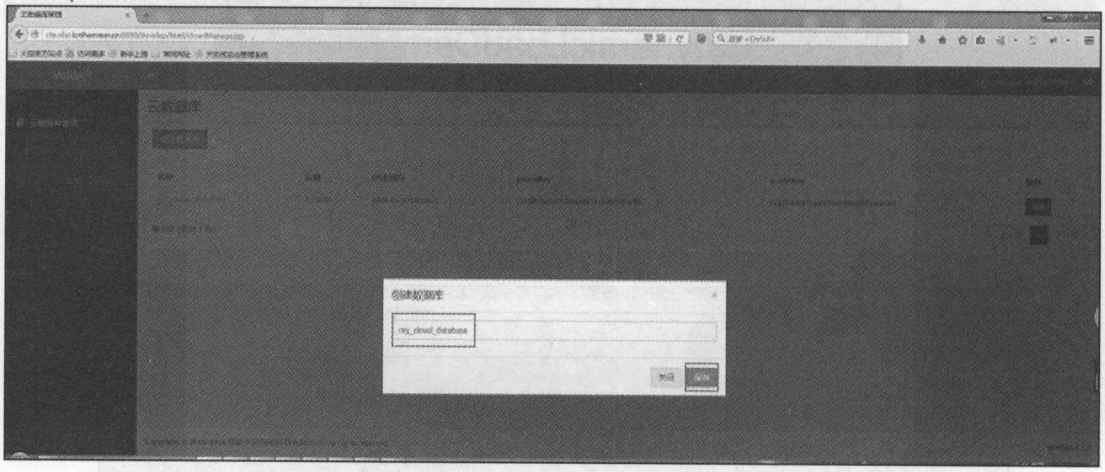
（2）登录。在浏览器打开 http://cloudar.ionhammer.cn:8080/develop/html/login.jsp 进行登录。



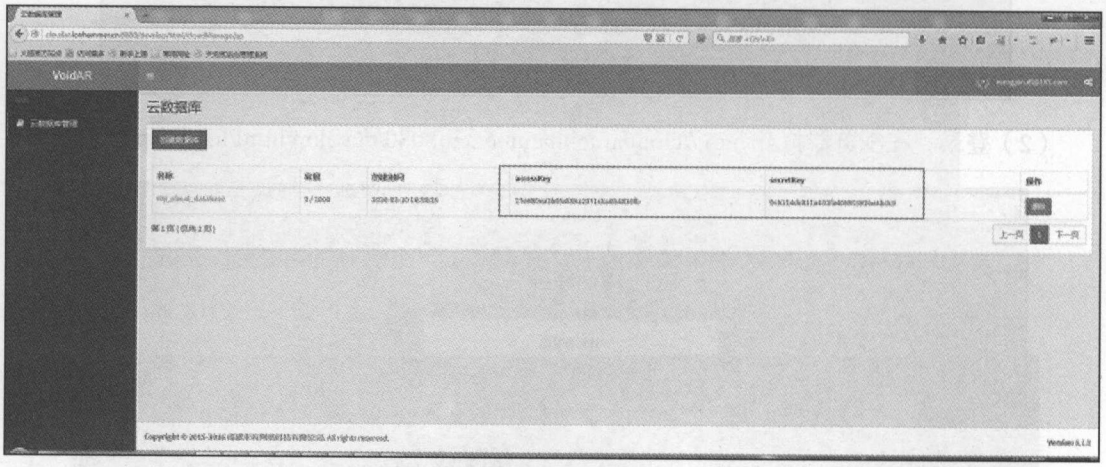
（3）创建云数据库。点击左上角的“创建数据库”按钮。



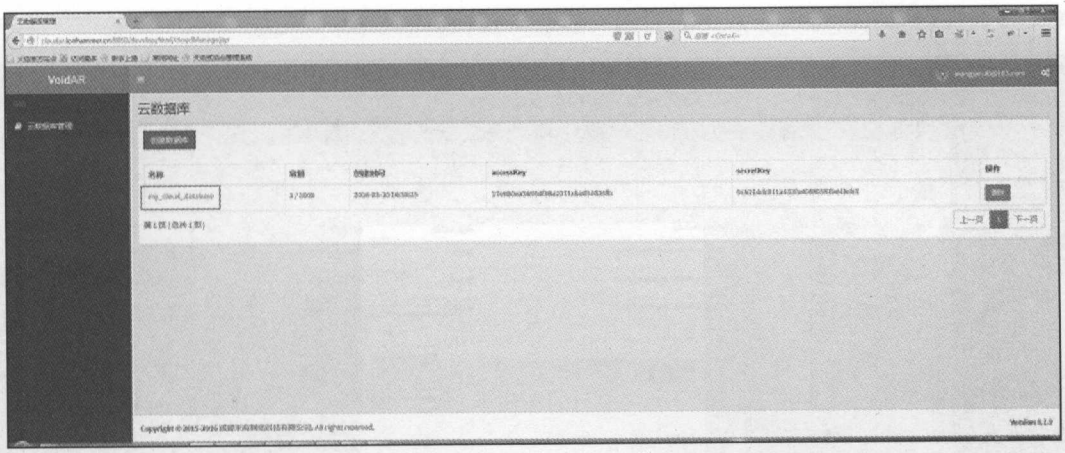
填写“数据库名称”，点击右下角的“保存”按钮进行创建。



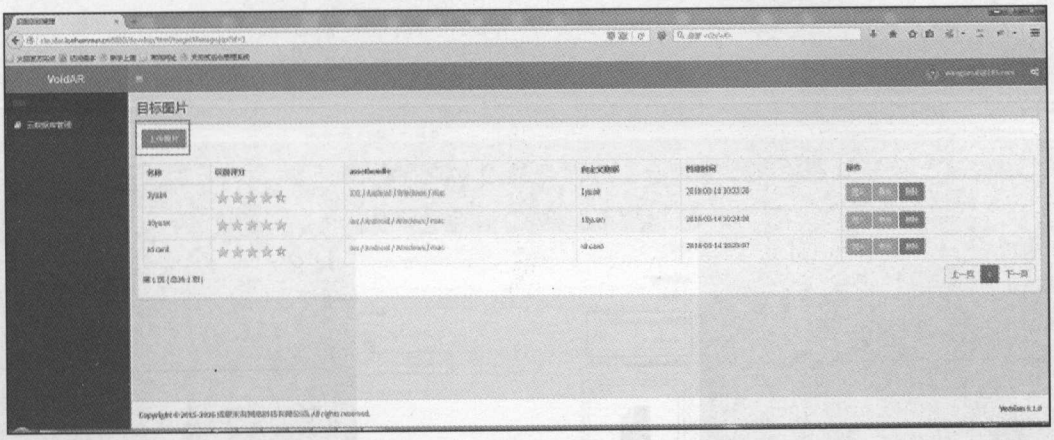
创建成功后，可以得到 accessKey 和 secretKey 信息，用于客户端访问。



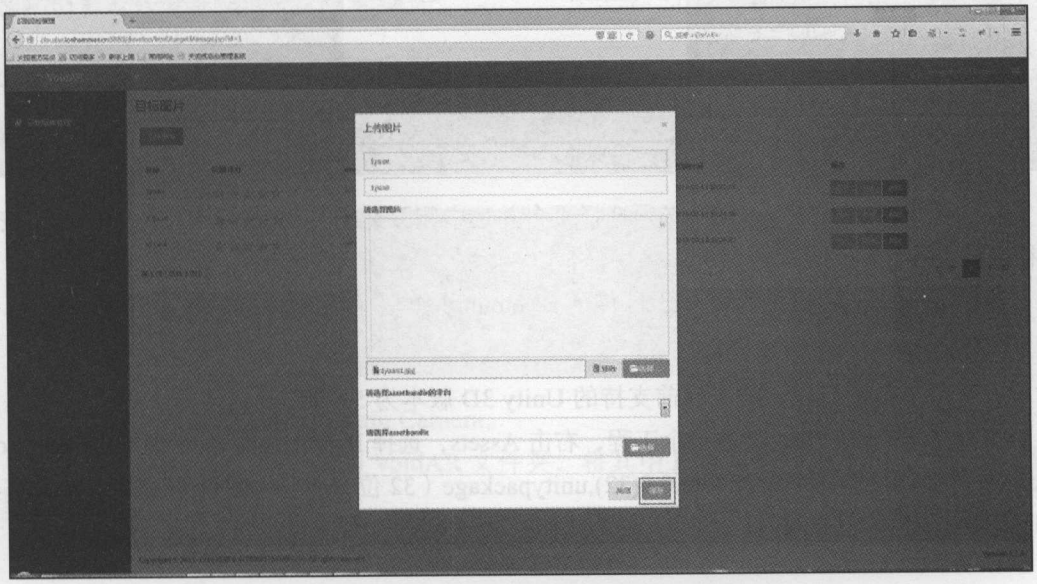
（4）上传识别图片。点击左边的云数据库名称。



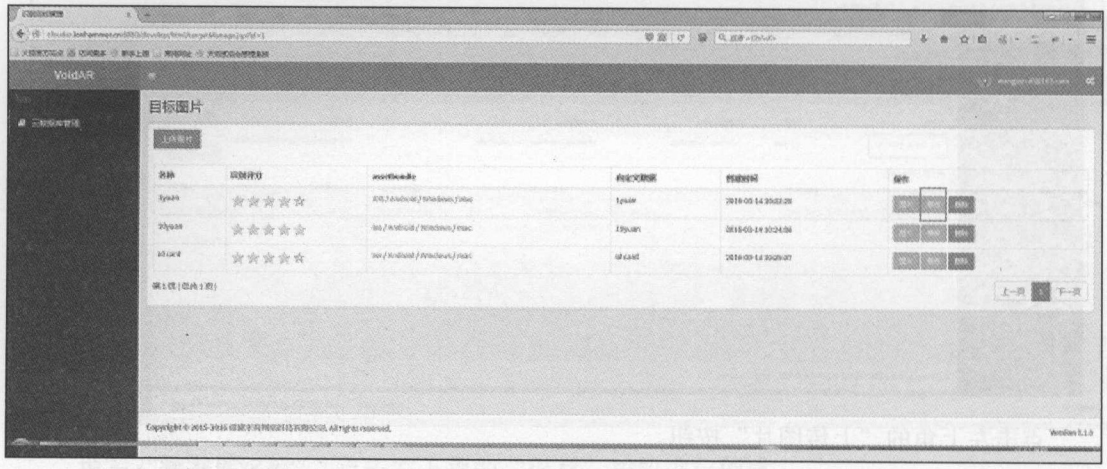
点击左上角的“上传图片”按钮。



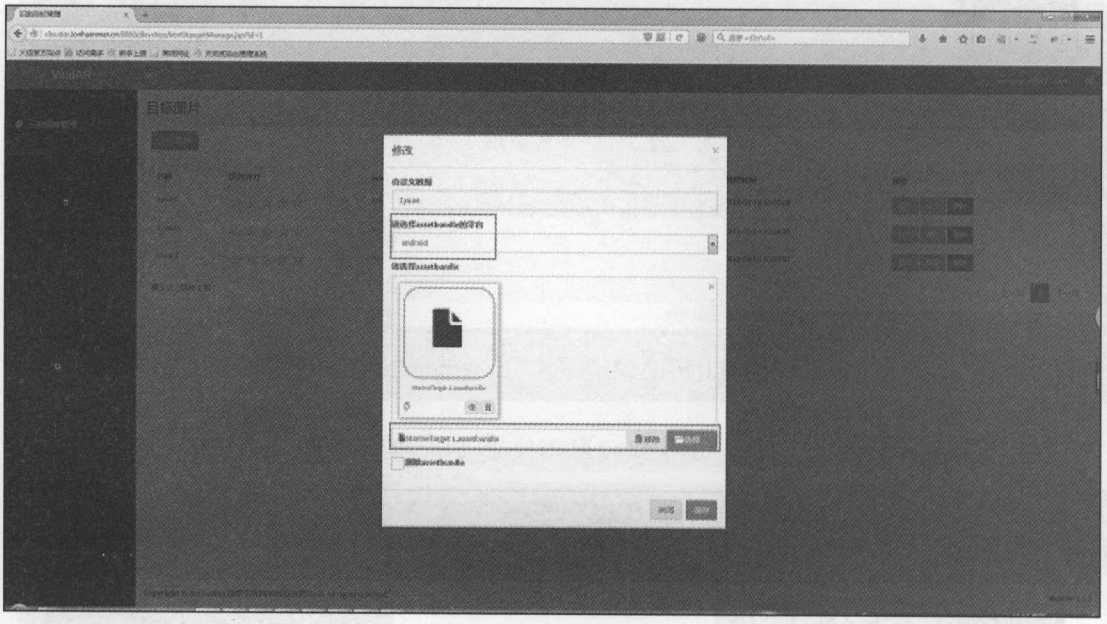
填写相关图片信息并选择要上传的识别图片（assetbundle 相关项可不填写），点击右下角的“保存”按钮进行上传。



（5）管理 assetbundle。点击右边的“修改”按钮。



填写 assetbundle 相关数据（必须选择平台和上传文件），点击右下角的“保存”按钮进行上传。



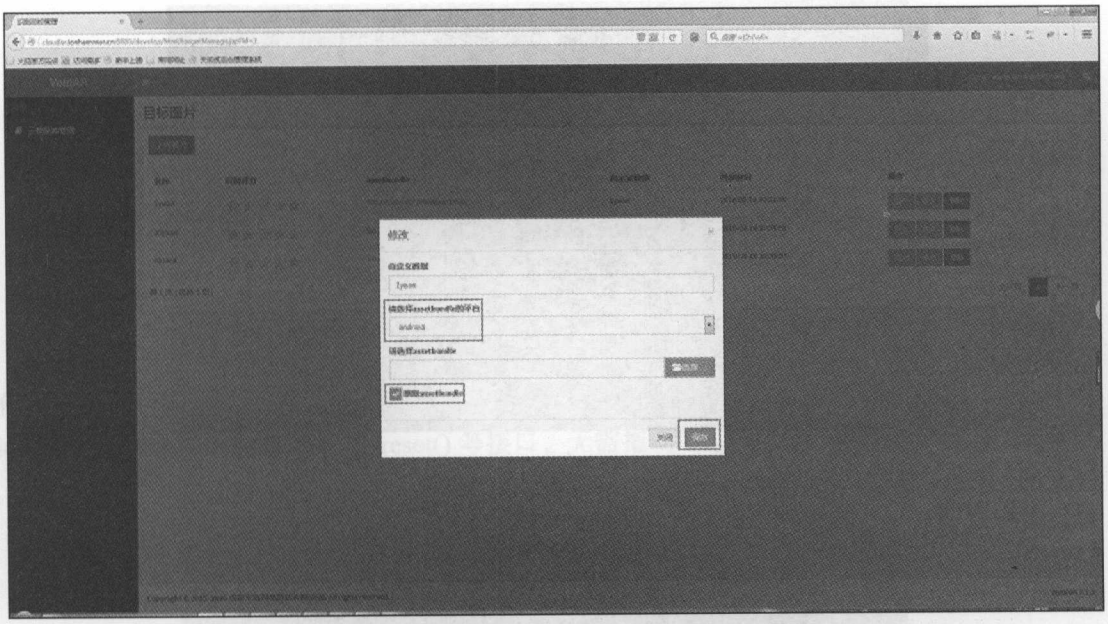
删除各平台的 assetbundle（必须选择平台并选中删除选项），点击右下角的“保存”按钮进行上传。

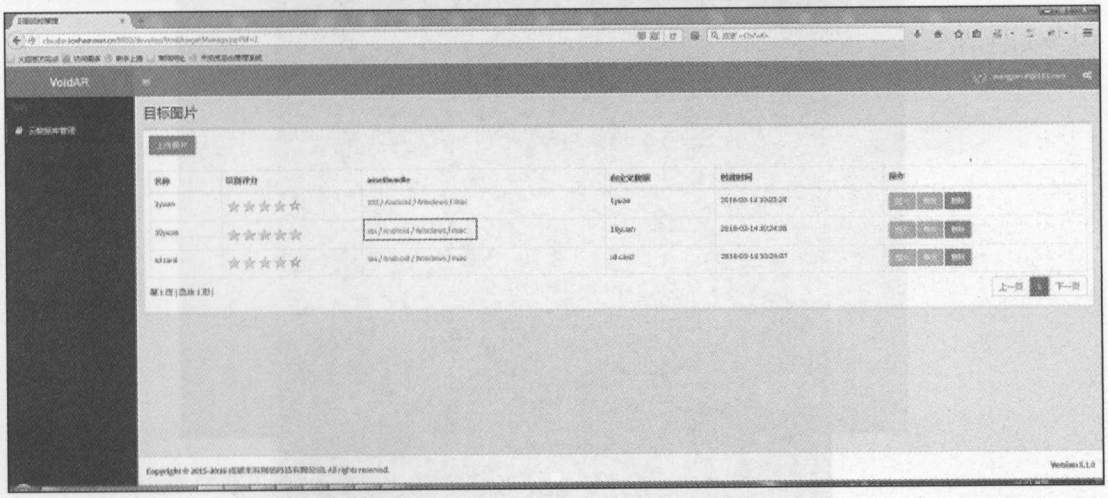
assetbundle 平台相关信息：已上 传了 assetbundle 的平台显示为绿色，没有则为灰色。

7.ShapeTarget 使用说明

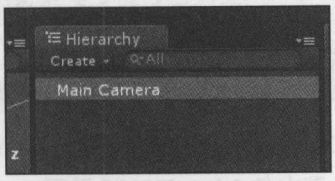
（1）首先安装 Unity 3D，目前支持的 Unity 3D 版本为 5.x 版。

（2）打开 Unity 3D，新建一个工程，右击 Assets，选择 Import Package，继续选择 Custom Package，随后选择 VoidAR-v0.0.8(32).unitypackage（32 位 Unity）或者 VoidAR-v0.0.8(64).unitypackage（64 位 Unity）。



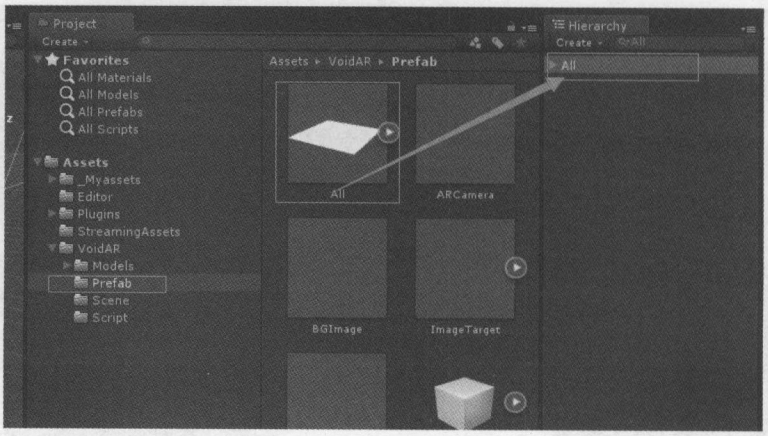


（3）在 Unity 3D 中点击 File 按钮选择 New Scene 创建一个新的场景，新场景自带了一个 Main Camera 。

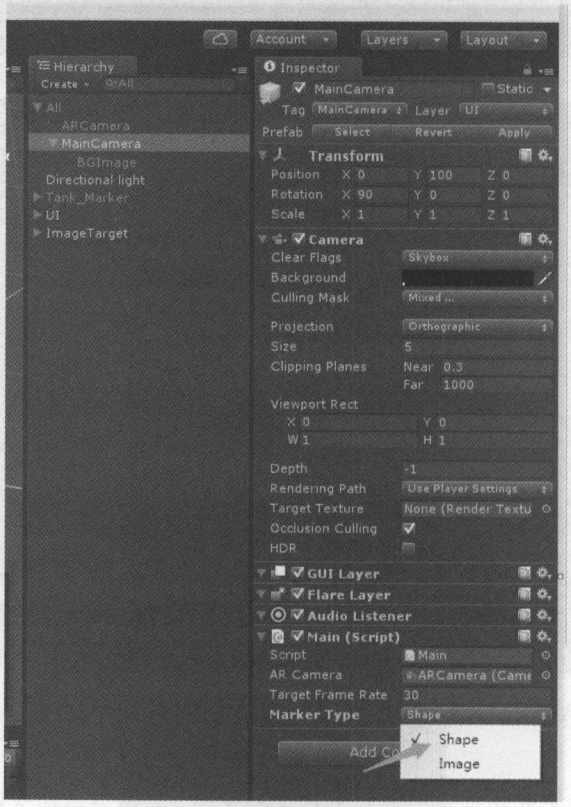


（4）删除新场景中自带的 Main Camera。

（5）在 Assets 文件夹下找到 VoidAR 文件夹，将其中 Prefab 文件夹下的“ ALL”预制件拖到场景中。



在“All”预制件中的 MainCamera 上，Marker Type 的选项 Image 表示图片识别，Shape 表示手绘识别，目前暂不支持 Shape 和 Image 混用。

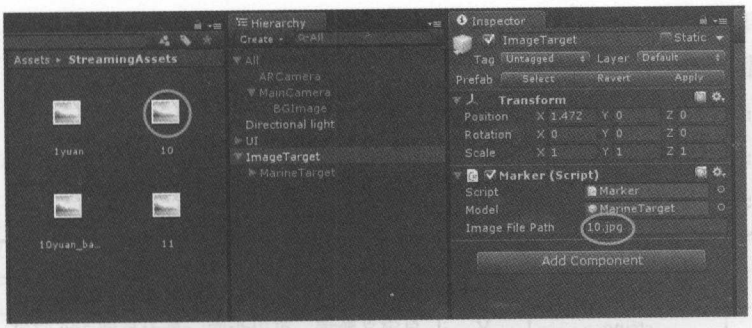


将 Prefab 文件下的“ImageTarget”预制件拖到场景中。

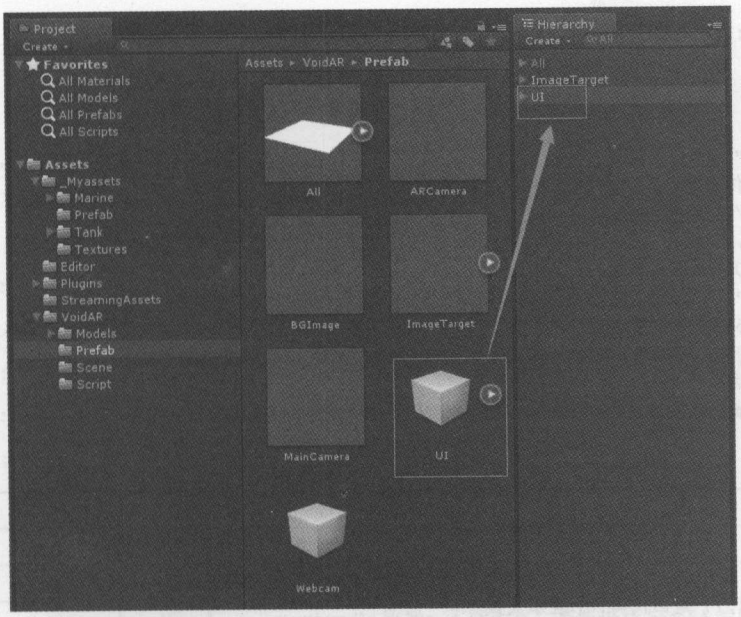
将你要显示的模型拖动到场景中的 ImageTarget 里面。

将刚才拖入 ImageTarget 里面的模型拖动到 Marker 的 model 里面。

在 Image File Path 里面填写图片的名字，例如 10.jpg（此版本暂时需要手动填写），目前版本图片必须放在 StreamingAssets 文件夹下。



（6）将 Prefab 文件下的“ UI”拖动到场景中，这里的 UI 主要是两个按钮，一个按钮的作用是添加目标，另一个按钮的作用是清空添加的所有目标。用户也可自定义 UI 自己调用 addCurrentShapeImageTarget ()、reset() 等接口，无需拖动 UI。



（7）点击运行就可测试。

注意：

在点击“添加目标”时，必须垂直正对目标，这样效果最好！

8. 云识别的使用

文档说明：使用接口前需要先到开发者平台进行如下操作。

- 注册（http://cloudar.ionhammer.cn:8080/develop/html/register.jsp）。
- 登录（http://cloudar.ionhammer.cn:8080/develop/html/login.jsp）。
- 新建云数据库，取得该数据库的 accessKey 和 secretKey。

接口说明：接口均使用 http post 方式访问，请求参数和返回结果均为 json 格式，并且需要在 http header 中增加“ Authorizationg”项进行访问权限设置，格式为“ accessKey :

secretKey”，具体使用可参考接口演示程序。

（1）上传图片

接口说明：上传单张图片到指定的云数据库。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/uploadTarget.do。

请求参数：

名称	类型	必要	备注
name	string	Y	图片名称，不能为空，最大长度 50 字符
matadata	string	Y	自定义数据，可以为空，最大长度 200 字符
imagedata	string	Y	图片数据，不能为空，需要进行 base64 编码，大小最大为 2M

Json 数据：

```
{"name":"10.jpg","metadata":"dGVzdA","imagedata":"4AAQSkZJRgA..."}
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":" 上传成功 ",// 返回信息
"image":{// 图片详细信息
"id":3521,// 图片 ID
"name":"10.jpg",// 名称
"metadata":"test",// 自定义数据
"rating":5,// 识别度评分（1- 差，2- 较差，3- 中，4- 较好，5- 好）
"bundle":"0,0,0,0",// 是否绑定了 assetbundle (ios,android,windows,mac)
// 等于 0 表示没有绑定，大于 0 表示已绑定
"createTime":"2016-03-30 14:11:19"// 上传时间
}
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
"msg":" 图片上传失败 "// 失败描述
}
```

演示程序：AddTarget.java。

（2）取得图片列表

接口说明：从云数据库取得指定数量的图片信息。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/targetList.do。

请求参数：

名称	类型	必要	备注
start	int	Y	记录开始位置
num	int	Y	记录数量，最大值为 20

Json 数据：

```
{"start" :0," num" :10}
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":"",// 返回信息
"total":1,// 图片总数
"data":[{// 图片详细信息
"id":3521,// 图片 ID
"name":"10.jpg",// 名称
"metadata":"test",// 自定义数据
"rating":5,// 识别度评分（1- 差，2- 较差，3- 中，4- 较好，5- 好）
"bundle":"0,0,0,0",// 是否绑定了 assetbundle（ios,android,windows,mac）
// 等于 0 表示没有绑定，大于 0 表示已绑定
"createTime":"2016-03-30 14:11:19"// 上传时间
}]
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
"msg":" 请求数据传输错误 "// 失败描述
}
```

演示程序：TargetList.java。

（3）上传 assetbundle

接口说明：上传 Unity 打包的 assetbundle 文件到云数据库，并与指定图片绑定。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/uploadBundle.do。

请求参数：

名称	类型	必要	备注
id	long	Y	图片 ID
platform	int	Y	平台，1-ios，2-android，3-windows，4-mac
bundle	string	Y	assetbundle 数据，不能为空，需要进行 base64 编码，最大为 2M

Json 数据：

```
{"id" :3521," platform" :1," bundle" :" VW5pdHlXZ..." }
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":"assetbundle 上传成功 "// 返回信息
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
```

```
"msg":" 请求数据传输错误 "// 失败描述
}
```

演示程序：UploadBundle.java。

（4）取得图片详细信息

接口说明：从云数据库取得指定图片的信息。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/targetDetail.do。

请求参数：

名称	类型	必要	备注
id	long	Y	图片 ID

Json 数据：

```
{"id" :3521}
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":"",// 返回信息
"image":{// 图片详细信息
"id":3521,// 图片 ID
"name":"10.jpg",// 名称
"metadata":"test",// 自定义数据
"rating":5,// 识别度评分（1- 差，2- 较差，3- 中，4- 较好，5- 好）
"bundle":"2,0,0,0",// 是否绑定了 assetbundle（ios,android,windows,mac）
// 等于 0 表示没有绑定，大于 0 表示已绑定
"createTime":"2016-03-30 14:11:19"// 上传时间
}
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
"msg":" 请求数据传输错误 "// 失败描述
}
```

演示程序：TargetDetail.java。

（5）删除图片

接口说明：从云数据库删除指定图片。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/deleteTarget.do。

请求参数：

名称	类型	必要	备注
id	long	Y	图片 ID

Json 数据：

```
{"id" :3521}
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":" 删除成功 "// 返回信息
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
"msg":" 请求数据传输错误 "// 失败描述
}
```

演示程序：DelTarget.java。

（6）删除 assetbundle

接口说明：删除指定图片绑定的 assetbundle。

接口地址：http://cloudar.ionhammer.cn:8080/develop/ws/deleteBundle.do。

请求参数：

名称	类型	必要	备注
id	long	Y	图片 ID
platform	int	Y	平台，1-ios，2-android，3-windows，4-mac

Json 数据：

```
{"id" :3521," platform" :1}
```

返回结果：

成功：

```
{
"code":"0",// 返回代码，0 表示成功
"msg":" 删除 assetbundle 成功 "// 返回信息
}
```

失败：

```
{
"code":"1",// 返回代码，1 表示失败
"msg":" 请求数据传输错误 "// 失败描述
}
```

演示程序：DelBundle.java。

（7）code 说明

code	备注	code	备注
0	成功	401	权限错误，accessKey、secretKey 错误
1	失败	999	系统错误

演示程序使用说明：安装 JDK1.7 和 Eclipse，然后将演示工程导入 Eclipse（File → Import... → General → Existing Projects into Workspace → Next）即可运行。

第12章 AR 创建工具：找趣 (realcast) 浏览器

12.1 什么是找趣

找趣是由 Realmax 打造的一款面向用户的增强现实制作平台与操作系统，致力于为使用者提供高效简便的 AR 制作环境和统一的 AR 操作系统入口。

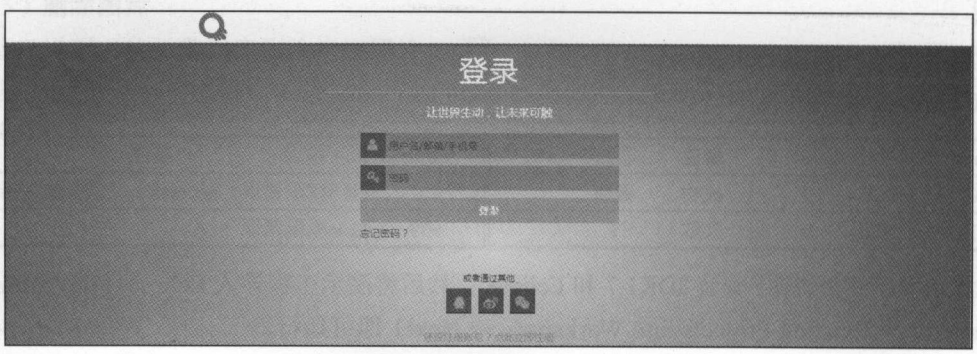
找趣旨在提供简单易用、功能完善的 AR 创作服务，包括 AR 制作管理后台、3D 图形化编辑器、AR 浏览器和 AR 创意商城；同时，找趣开放了众多企业通用的项目模板以及定制化的 AR 解决方案。

12.2 找趣后台

找趣后台是找趣云平台的重要组成部分，承载了找趣的大部分功能，旨在建立一个一站式的管理和服务中心。

1. 注册 / 登录 / 注销

在登录页面内，输入正确的邮箱 / 手机号、密码和随机验证码，点击“登录”按钮。



登录页面内，选择 QQ/ 新浪微博 / 微信的方式，在打开的页面中输入账号、密码，进行快速登录。



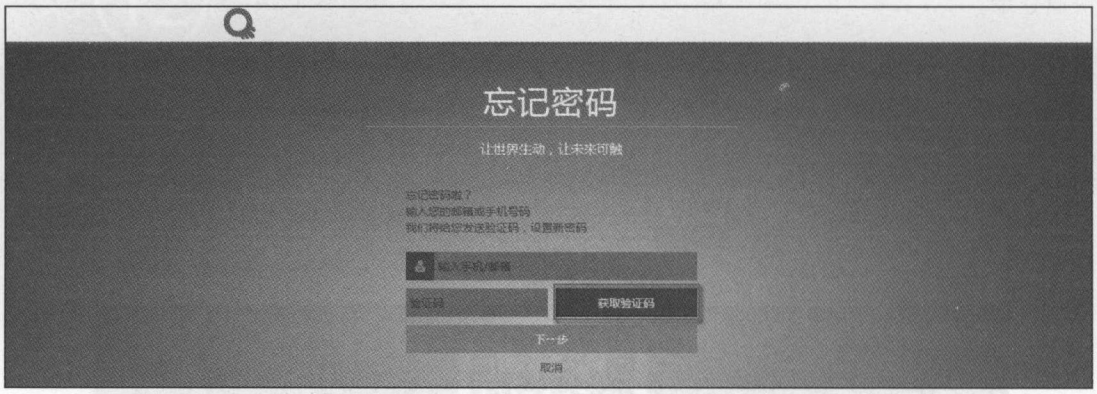
登录成功后，系统将自动跳转到后台首页。

2. 重置密码

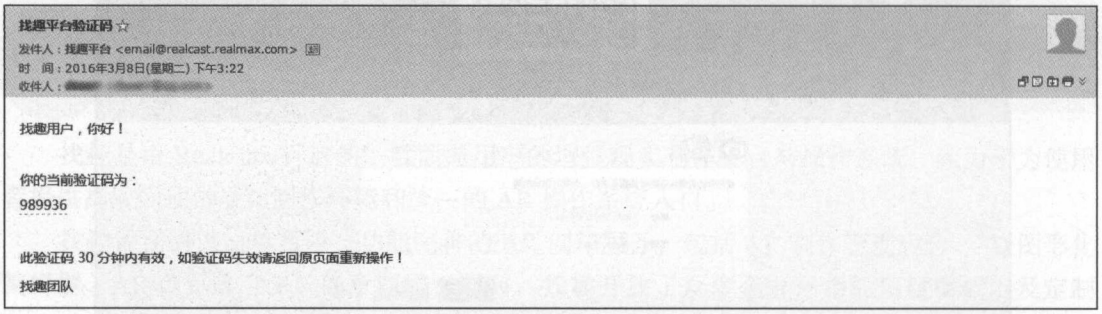
如果无法访问账户，可以重新设置密码。在登录页面下方，点击“忘记密码?”并前往重置页面。



输入电子邮箱或者手机号码，点击“获取验证码”按钮。



系统会向你的邮箱 / 手机号中发送一封包含验证码的邮件 / 短信。



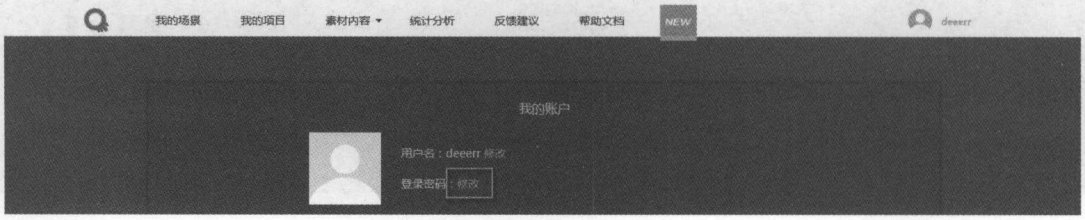
返回登录页面，将得到的验证码填入验证码框，并输入新密码，点击“确认”按钮即可重置密码，重置成功后系统会自动跳转回登录页面。

3. 修改密码

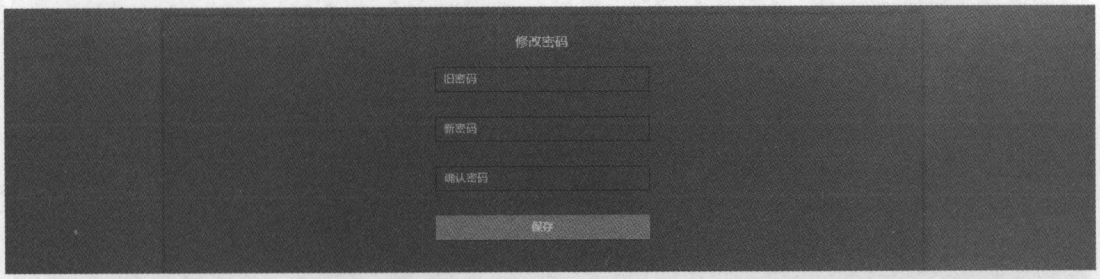
登录找趣后台，点击界面右上角的账户名，在弹出的菜单中选择“我的账户”以进入账户信息界面。



在登录密码右侧点击“修改”。

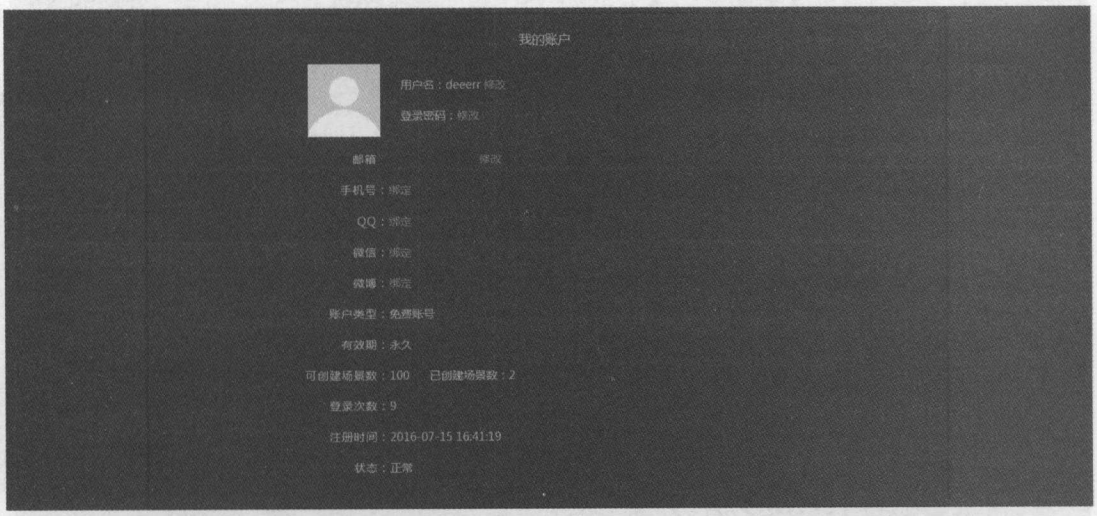


在弹出的页面中输入正确的旧密码以及两次新密码，保存后完成密码修改。



4. 账户信息设置

登录找趣后台，点击界面右上角的账户名，在弹出的菜单中选择“我的账户”以进入账户信息界面。在账户信息页面中可以看到当前账户的一些基本信息。



点击左上角的图片可上传一张头像；点击邮箱右边的“修改”可更换登录邮箱；绑定手机号、QQ、微博账户，以便下次直接使用三方登录。

12.3 创建场景

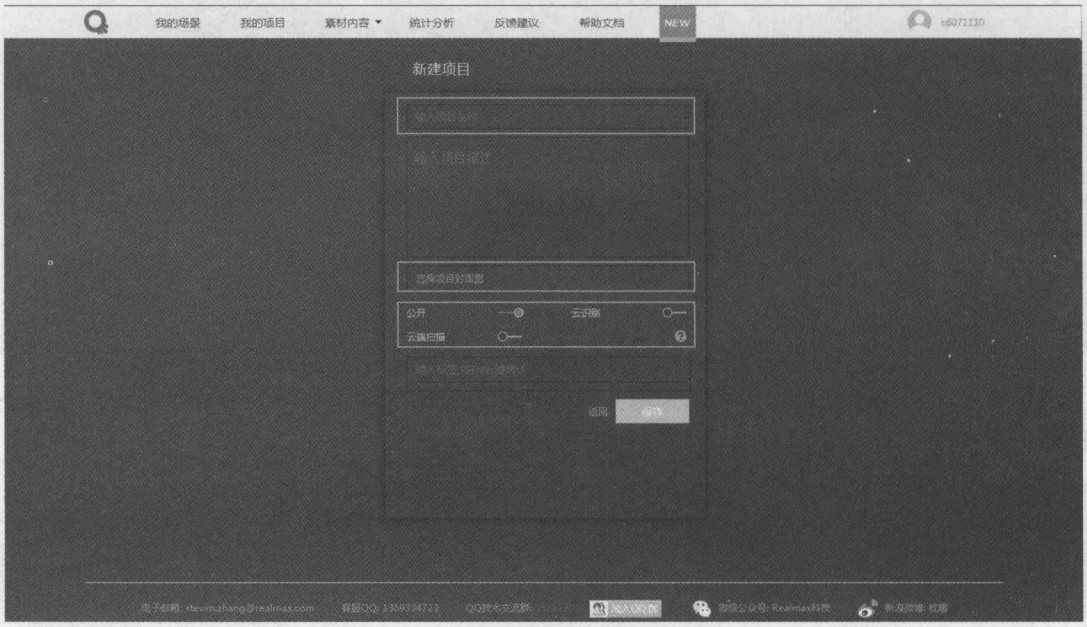
创建场景需要从创建项目开始。首次登录后，系统会新增一个默认项目，用户可以在默认项目中添加场景，也可以选择新建一个自定义项目。

在我的项目页面选择“新建项目”，将会出现以下界面。

（1）基本信息：输入项目名（*）、项目封面图（*）、标签和项目描述。

（2）设置。





- 公开：公开的项目将会出现在找趣 App 页面内，其他用户可以搜索到该项目并进入查看。
- 云识别：打开后，项目下可容纳千万张识别图，可以快速打开项目；关闭后为离线识别，项目内最多容纳 100 张识别图，且识别图越多，打开项目越慢。
- 云端扫描：启用后，可以使用找趣 App 的云端扫描识别封面图、进入项目（效果等同于频道二维码），此功能须经管理员审核。

保存后，系统自动跳转至“我的项目”页面。此时，可前往“我的场景”页面开始创建场景。

12.4 制作场景

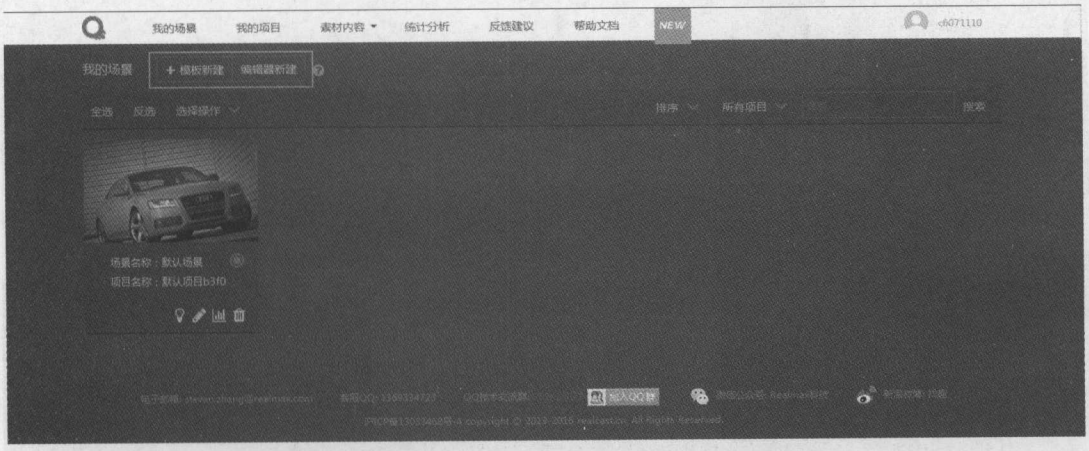
1. 准备

一个完整的场景需要包含下面两部分素材：

- 识别图：AR 对象的载体，基于二维图片的识别要求上传一张符合规范的图片，格式为 .jpg。
- AR 内容：将要呈现出来的 AR 对象，类型为模型和图片，需要将 AR 对象转换成找趣要求的格式。

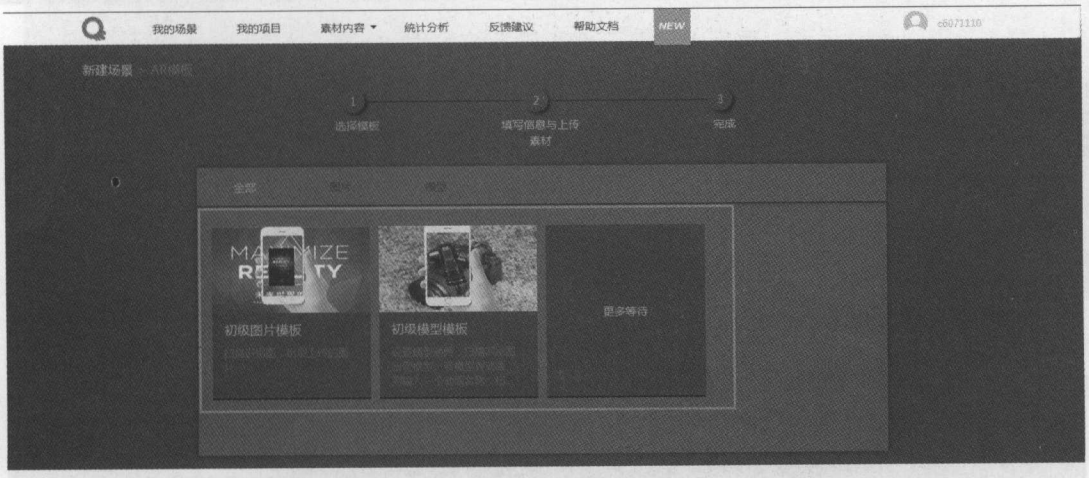
2. 制作

登录找趣云平台，在我的场景页面下内选择“模板新建”或者“编辑器新建”，进入制作界面。

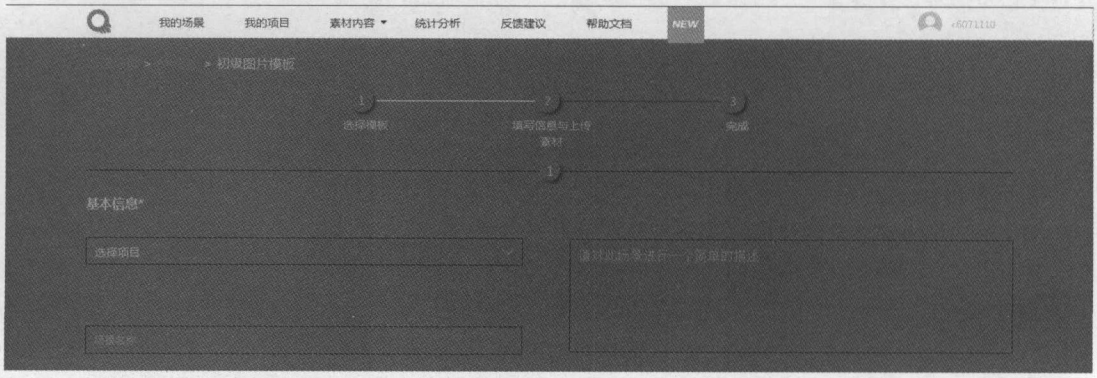


3. 模板新建

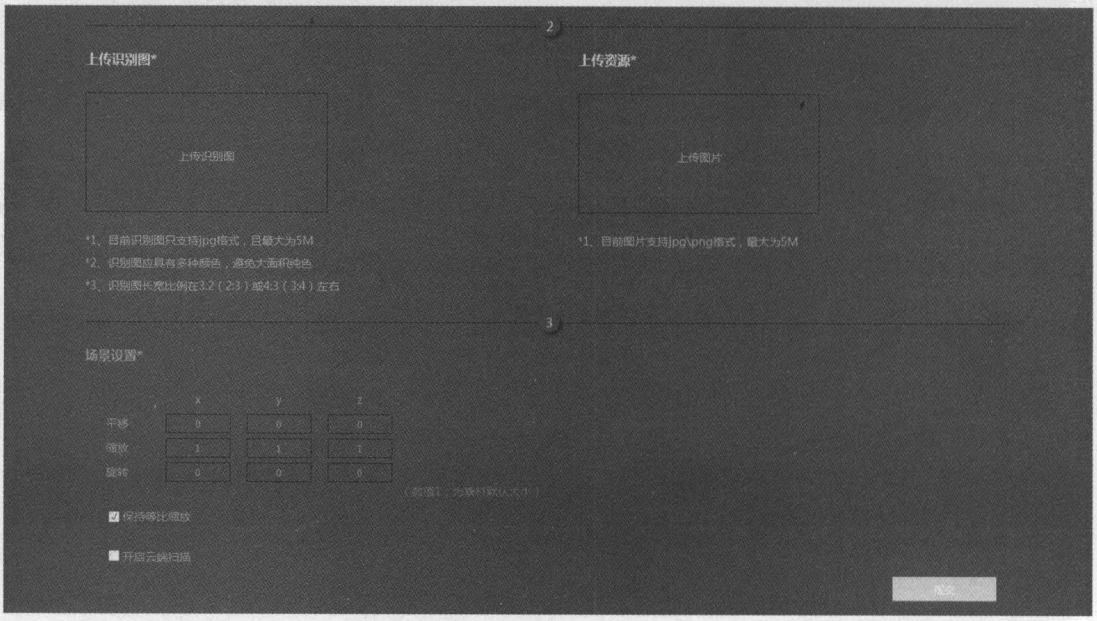
点击模板新建后进入模板选择页面，在提供的列表里点击一个合适的模板。



按照顺序输入场景信息（所属项目、名称和描述）。



上传识别图和 AR 对象（资源）。修改场景设置，如不需要调整，则空置。

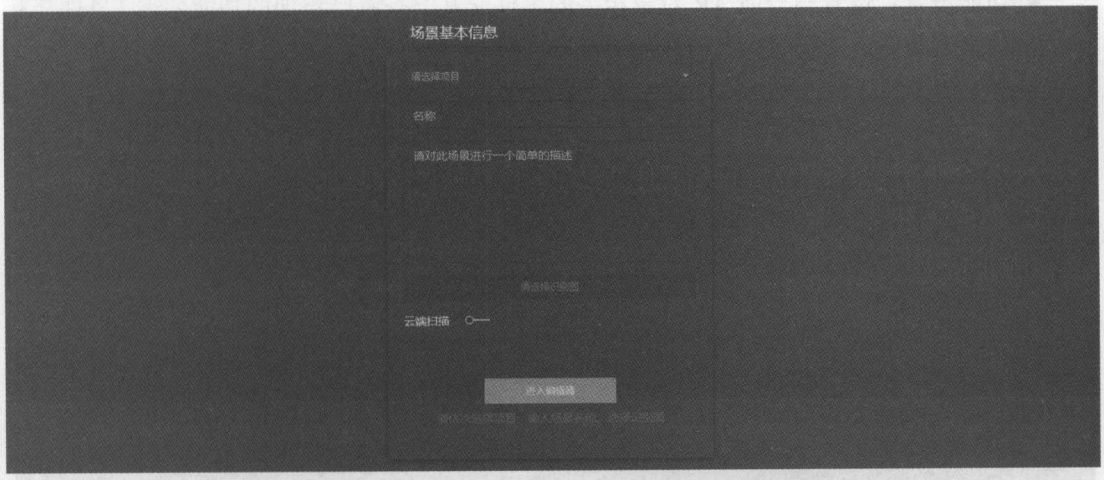


提交后即可生成场景，页面上会显示查看场景的方法。



4. 编辑器新建

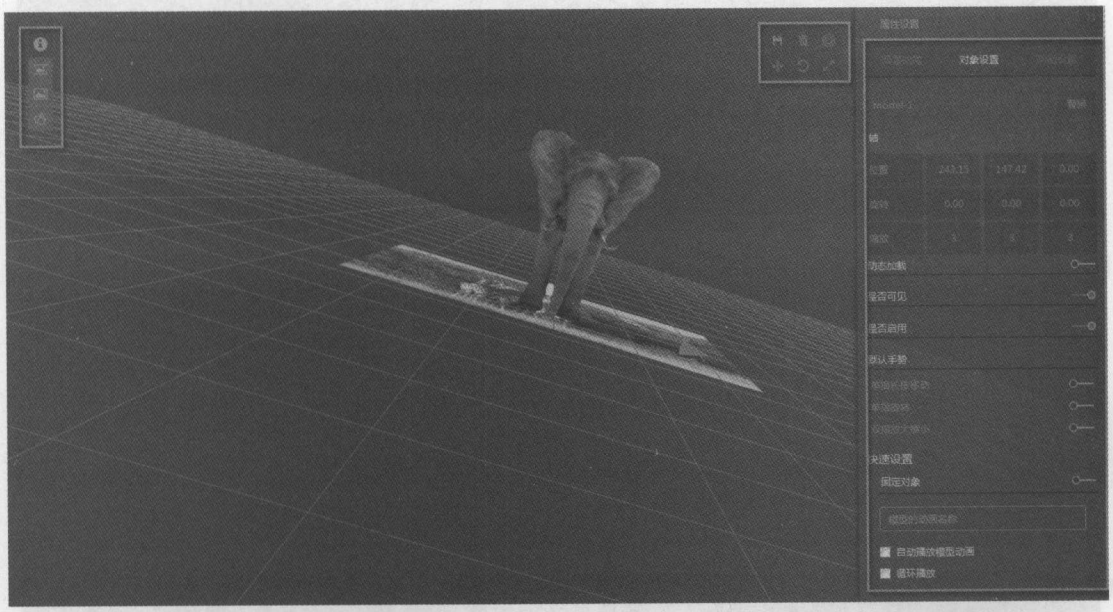
点击 AR 编辑器后进入场景信息页面，填写场景的属性和识别图。



（1）基本信息：所属项目、场景名称和场景描述。

（2）识别图：选择一张 .jpg 格式的图片作为 AR 识别图。“分享到云端扫描”决定了是否能够通过找趣 App 的云端识别界面直接扫描图片、查看该 AR 对象。

保存后进入编辑器页面。左上角可选择更换识别图 / 增加模型 / 图片，等候加载成功。在编辑框内可调整 AR 对象与识别图的相对位置。



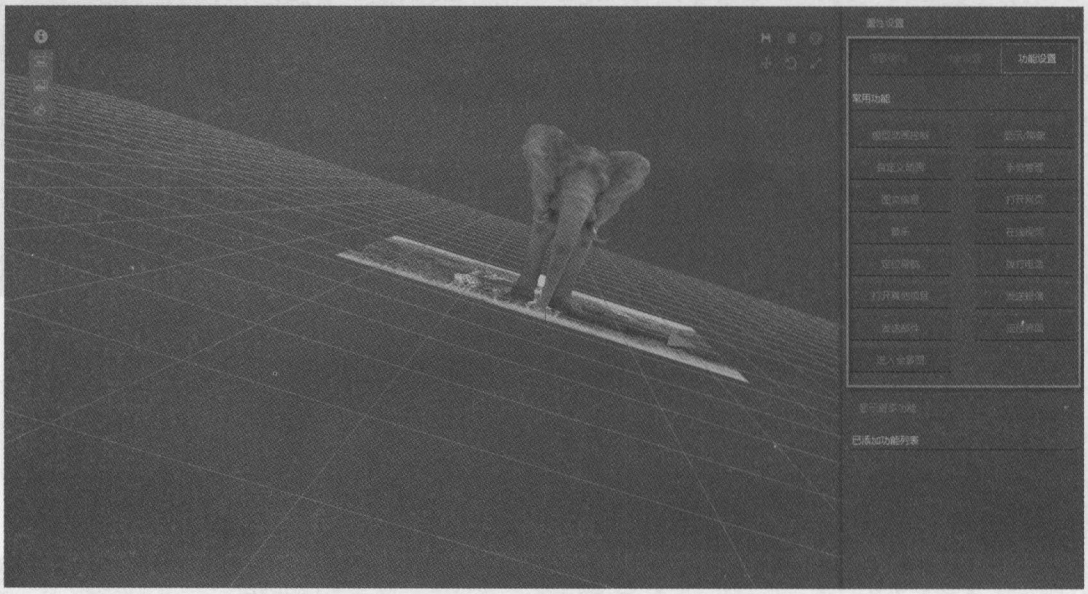
右边属性列表内有三个切换选项，分别为：

- 场景结构：显示场景基本信息（允许修改是否分享到云端）；同项目下不同场景的结构图，允许直接点击场景进行切换编辑（切换前请确定已保存对当前场景的修改）。

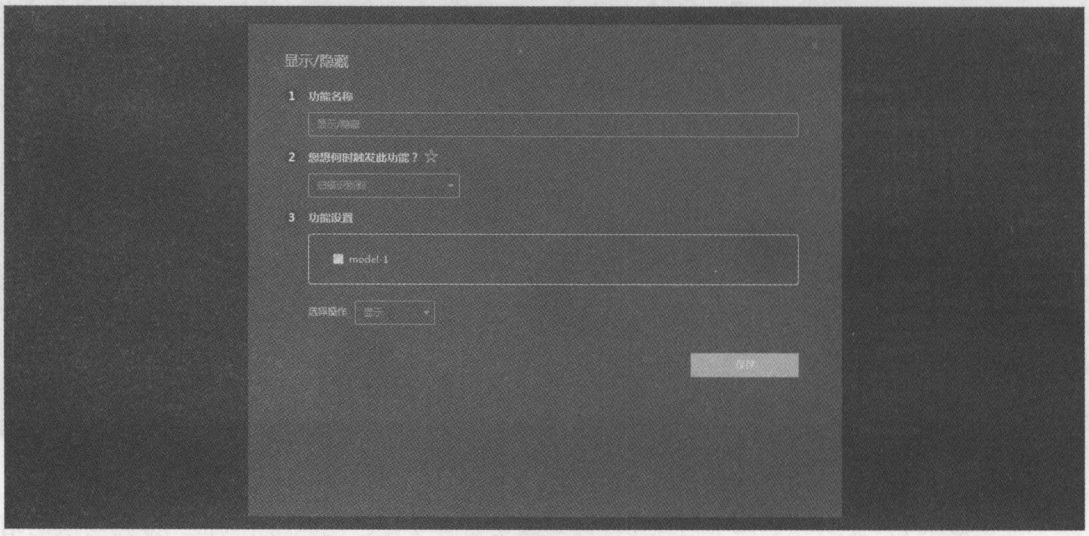
修改 AR 内容的属性设定，包括名称、是否可见、状态参数、加载类型、状态和手势。

- 对象设置：修改 AR 对象的属性设定，包括名称、替换当前对象、相对位置、动态加载、是否可见、是否启用、默认手势以及对象相关的快速设置。
- 功能设置：设置与当前 AR 对象相关的事件交互，查看已添加的功能列表（事件交互的具体功能定义在后面章节有详细解释说明）。

素材上传完成后，开始给 AR 对象设置事件交互。选中某一对象，右侧属性列表切换至功能设置，选择一个功能。



在弹出的窗口中输入功能相关的设置。选择触发时机，若希望扫描到时就发生，则直接选择“扫描到”“扫描丢失”事件。某一特定 AR 对象的特定事件则可在时机列表中拉选。



按照需求依次编辑完成后，点击“保存”按钮，制作过程结束。

12.5 如何使用找趣

找趣浏览器提供了大量的展示项目，可以让 AR 新手快速了解 AR、喜欢 AR。对找趣用户而言，它还是一个超级管家，你的每一次灵光闪现都将被记录下来，如果你愿意，还可以将它展现给所有找趣用户，轻松搭建用户间的交流通道。

1. 玩法说明

（1）云端扫描：找趣提供直接识别的形式，即刻扫描图片，就能查看 AR 效果（识别图）或者打开对应的项目（项目封面图）。需要将识别图或封面图分享到云端。

（2）热门推荐：页面下提供了大量的项目入口，包括官方推荐和用户的热门频道，只要点击图片，就能快速打开、查看项目内的场景。项目和项目下的识别图都可以进行分享。

2. 查看说明

使用找趣查看 AR 效果分为三步：

（1）准备另一台设备，打开识别图；或者将识别图打印出来（不要使用反光或者有漫射材质的纸张打印）。

（2）打开找趣 App，进入对应的项目。

（3）待 App 准备好后，将手机摄像头对准识别图，即刻体验 AR。

3. 怎样打开频道?

在移动端找趣 App 上打开频道有四种不同的方式：

（1）在云端扫描模式下直接识别项目封面图：这种方式需要用户在创建项目时打开云端扫描属性。

（2）扫描项目二维码：在后台的项目页面下，每个项目都对应一个独一无二的二维码，使用找趣 App 提供的二维码扫描工具扫描，便会打开该项目。

（3）按项目名搜索：在最上方的搜索框内输入项目名称，点击列表内的项目即可打开。

（4）登录后查看：使用找趣账号登录浏览器，进入“我的记录”，在我的页面可以看到账号下的项目，同时还可查看和打开收藏的频道。

12.6 管理场景 / 项目

1. 管理场景

在“我的场景”页面下可以看到用户制作的所有场景，如果拥有多个项目，则在右方的项目列表里选择所属的项目，或者在搜索框内输入名称。

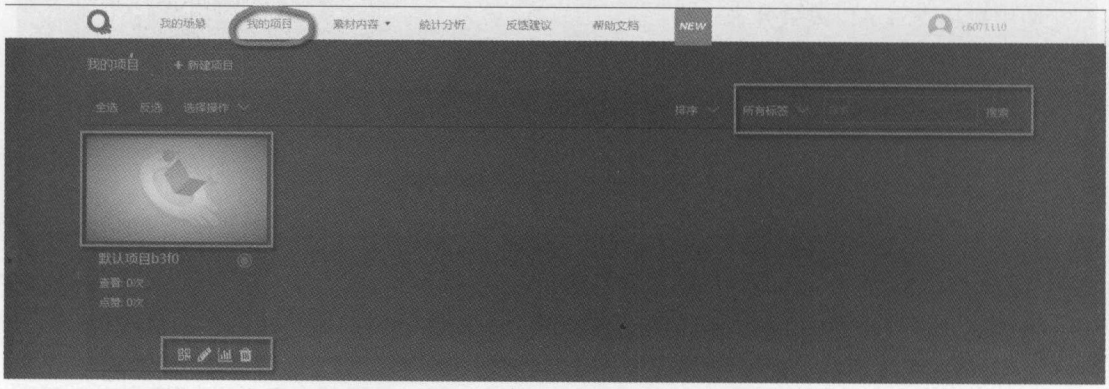
点击场景图片可以打开所属项目二维码以及场景识别图。

每个场景下方都有四个功能按钮，分别为禁用 / 启用、编辑、数据统计和删除操作。再次编辑不同制作方式下的场景时，依然打开原始制作状态下的编辑页面。



2. 管理项目

在“我的项目”页面下可以看到用户创建的所有项目。如果拥有多个项目，则在右方的标签栏里选择项目对应的标签，或者在搜索框内输入名称。

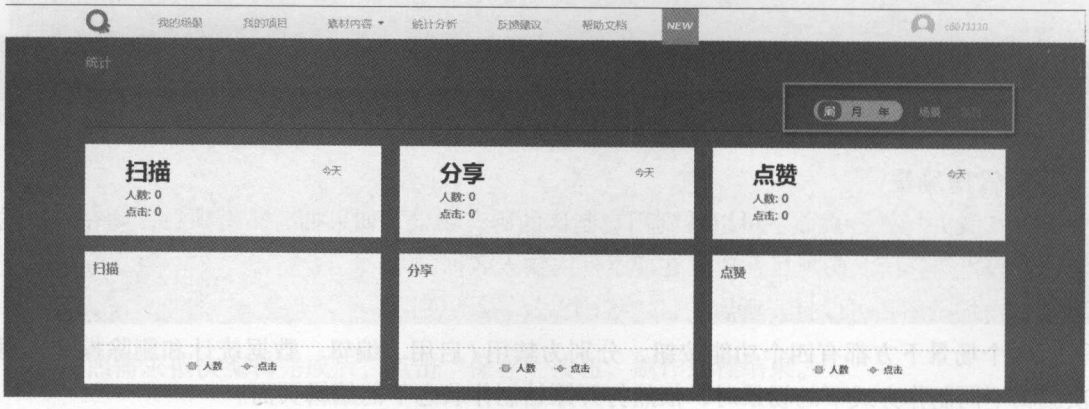


点击项目图片可以查看项目详细信息以及项目下的场景识别图。

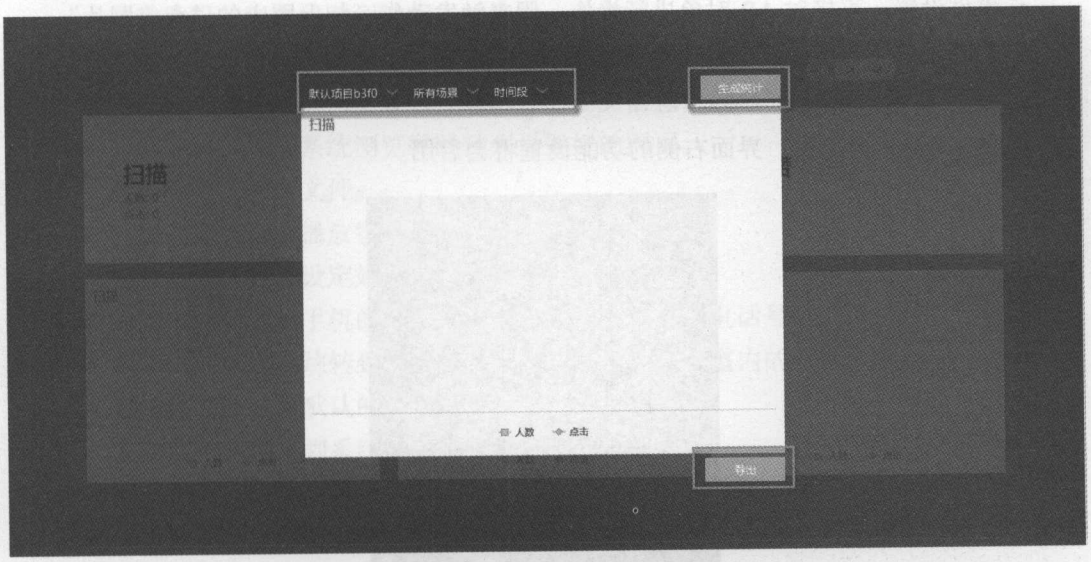
每个项目下方都有四个功能按钮，分别为查看频道二维码、编辑频道的详细信息、数据统计和删除操作。

3. 数据统计

对场景和项目的查看次数进行统计分析，能够很好地帮助用户探寻 AR 更深层的价值。



找趣云平台提供了场景和项目的扫描、分享和点赞数据统计，允许自定义对象和时间区域以生成统计报表，并可以导出 · png 格式的文件。



12.7 事件交互的定义

事件交互是 AR 编辑器最突出的新功能之一。找趣提供了一系列常用的功能选项，用户可以依据自己的需求来定义功能被触发的时机。

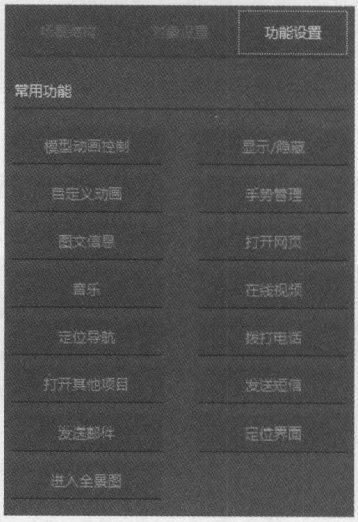


上面是一个交互式场景的示例：识别到 AR 对象以后，可以点击 AR 对象来打开网页。一个完整的“事件交互”包含两个部分：

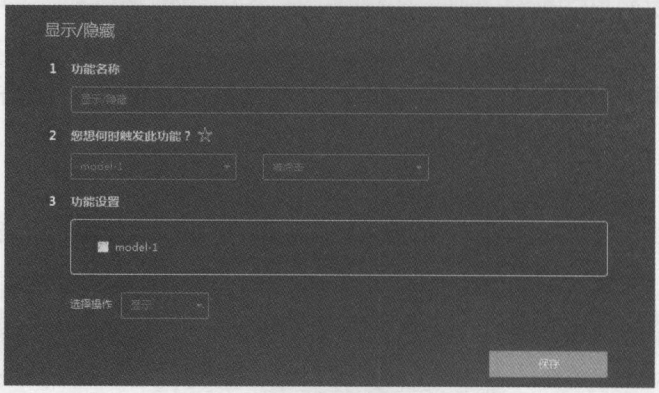
- 事件设置：直接对 AR 对象进行操作，用来触发动作，如上图中的“点击图片”。
- 功能设置：触发后反馈另一个交互，如上图中的“打开网页”。

1. 事件交互设计

选中一个 AR 对象后，界面右侧的功能设置将会启用。



功能设置里罗列了一部分常用的交互功能，以“显示 / 隐藏”动作为示例，点中后会弹出如下的页面。



- 输入功能名称：名称是用户对当前功能的定义，用于标志当前动作。
- 设置动作触发时机：触发事件的设定，在这里要选定对象和事件。
- 选择功能实现的对象：在列表内勾选（可多选）对象，并选择是“显示”还是“隐藏”。

保存后，隐藏动作的事件交互就设置好了。

2. 常用功能（只解释主要属性和意义）

- 模型动画控制：对选定模型的选定动画进行打开或关闭操作。

- 显示 / 隐藏：显示 / 隐藏同一张识别图下的 AR 对象，可多选。
- 自定义动画：选定的 AR 内容按照设置好的起始点、结束点和动态效果进行移动，以某种效果的形式移动到结束点并持续一段时间。
- 手势管理：选定的 AR 内容允许 / 禁止某个或某几个手势操作。
- 图文信息：以网页的形式打开编辑好的图文信息，内容可自定义。
- 打开网页：使用系统浏览器跳转到设置好的网页。
- 音乐：播放音频文件。
- 在线视频：全屏播放设置好的在线视频。
- 定位导航：打开设定好的地图坐标。
- 拨打电话：调用手机的打电话功能，拨打设置好的电话号码。
- 打开其他频道：跳转到其他频道，可以扫描其他频道内的 AR。
- 发送邮件：打开默认的邮箱面板。
- 发送短信：跳转到系统的短信界面。
- 定位界面：设定后的 AR 对象在扫描丢失后依然会出现在屏幕上。
- 进入全景图：打开全景图片。

3. 事件（触发时机）说明

允许设置的触发时机类型如下：

- 扫描识别到：使用 App 扫描图片，出现模型的瞬间触发动作（该事件无需触发对象）。
- 扫描识别丢失：使用 App 扫描图片，丢失模型的瞬间触发动作（该事件无需触发对象）。
- 点击：用户使用 App 扫描图片、出现模型后，点击该模型以触发动作。
- 被显示 / 被隐藏：显示 / 隐藏功能发生完成后触发功能。
- 被移动 / 被旋转 / 被伸缩：用户移动 / 旋转 / 伸缩当前模型的时候，或者由于其他 AR 对象的事件导致当前模型变化的时候便会触发功能。
- 动态加载完毕：设定好的对象实时加载完毕后触发功能。
- 模型动画开始 / 暂停 / 停止：模型对象的动画操作发生变化时触发功能。

12.8 模型动画的播放

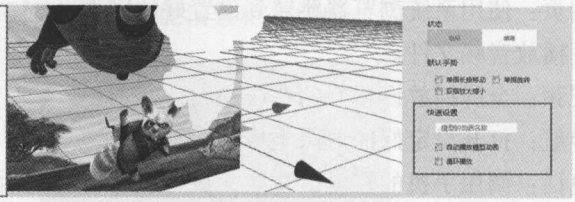
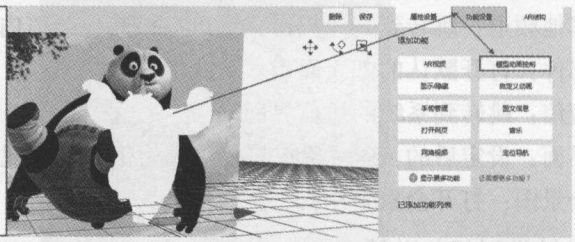
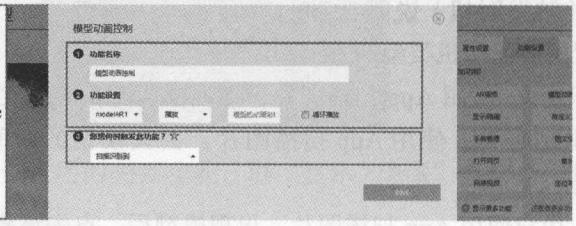
制作模型类 AR 的时候，常常会给模型添加一系列的动作使整个 AR 效果更加精彩。不过使用 AR 编辑器的时候，模型的动画可以通过互动来控制，这在设置上要复杂一些，但是 AR 的可玩性和互动性得到了极大的提高。

为了制作带有动画效果的模型，需要在 AR 内容属性列表中将动画添加至系统。未勾选“自动播放”的模型动画，可以由“动作：模型动画”来触发；另外，“动作：模型动画”也可以控制模型动画的暂停和停止。

注意：

找趣支持 .fbx 和 .obj 格式的模型，但 FBX 格式的文件在编辑器中无法看到贴图，使用 App 扫描识别时则无影响。

设置方法：

决定是否快速设置模型的动画播放效果。	
切换功能设置“动作：模型动画”。	
功能设置选择：模型名，动画操作，动画名（默认 Take 001），是否循环。决定触发时机。	

动作设置：模型动画的动作属性需要填写三个关键变量：模型对象；动画的播放 / 暂停 / 停止操作；输入需要播放的动画名称。

触发时机：任意 AR 内容的任意事件都可以作为触发事件。

注意：

添加模型动画时，动画的名字一定要与模型内置的英文动画名一致；模型动画暂停后，需要再次单独设置事件 – 动作来触发动画的开始播放，而不能通过原有事件来控制；由模型软件直接导出的 .fbx 格式文件会带有一个整体的动画名（Take 001），如果需要分割则需做相关处理。

到此，找趣浏览器的学习就告一段落了，更多有趣的功能还有待大家的进一步发现，希望大家可以发挥想象力，使用找趣定制出更炫更酷的 AR 应用。

第二部分

VR 开发

- 第 13 章　VR 技术简介
- 第 14 章　基于 Google Cardboard 开发 VR 应用
- 第 15 章　基于 HTC Vive 开发 VR 应用
- 第 16 章　基于 HTC Vive 实现房间漫游
- 第 17 章　基于 HTC Vive 开发投篮游戏
- 第 18 章　其他 VR 设备简介

第13章 VR技术简介

虚拟现实（Virtual Reality，VR）简称虚拟技术，也称虚拟环境，是利用电脑模拟产生一个三维空间的虚拟世界，提供关于视觉等感官的模拟，让用户感觉仿佛身临其境，可以及时、没有限制地观察三维空间内的事物。用户进行位置移动时，电脑可以立即进行复杂的运算，将精确的三维世界视频传回以产生临场感。该技术集成了计算机图形、计算机仿真、人工智能、感应、显示及网络并行处理等技术的最新发展成果，是一种由计算机技术辅助生成的高技术模拟系统。

在著名科幻作家刘慈欣的长篇小说《三体》中，有一套可穿戴设备叫作“V装具”，是《三体》游戏玩家所使用的设备，在小说中是这样描述这套设备的：

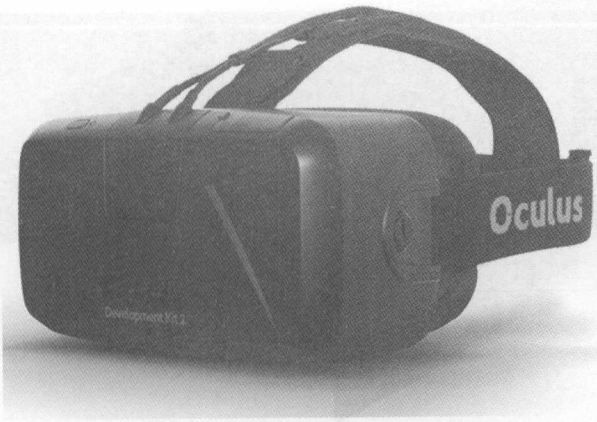
汪淼敲门，门没锁，开了一个缝，他看到申玉菲正坐在电脑前玩游戏，令汪淼惊奇的是她竟穿着一套“V装具”。这是目前在游戏玩家中很流行的玩意儿，由一个全视角显示头盔和一套感应服构成，感应服可以使玩家从肉体上感觉到游戏中的击打、刀刺和火烧，能产生出酷热和严寒，甚至还能逼真地模拟出身体暴露在风雪中的感觉。

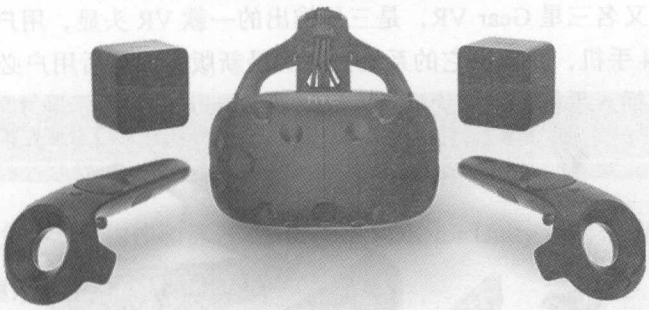
上述文字很好地描述了VR技术未来的发展方向，VR不但能给用户带来视觉冲击，还应当能给用户带来触觉、嗅觉等其他感官的全新体验。从目前VR设备的发展来看，我们只是解决了其中很小的一部分问题，未来还具有很大的发展空间。

近几年，VR技术火速发展，并诞生了很多VR相关硬件以及设备，这些设备主要分为主机设备和移动设备，主机设备以以下三种设备最受关注。

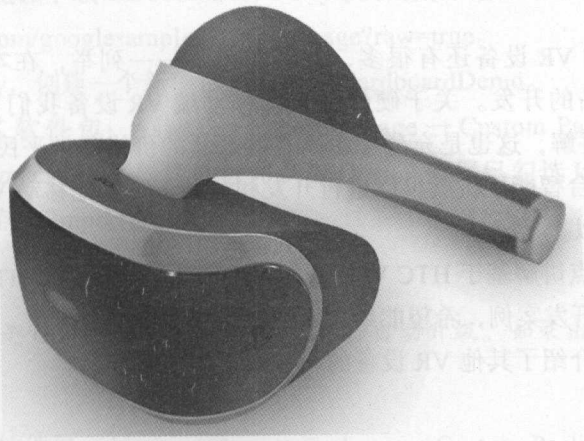
（1）Oculus是Facebook在2014年7月宣布以20亿美元的价格收购的VR设备制作商，在Facebook看来，Oculus的技术开辟了全新的体验和可能性，不仅在游戏领域，还在生活、教育、医疗等诸多领域拥有广阔的想象空间。

（2）HTC Vive是由HTC与Valve联合开发的一款VR头显产品，于2015年3月在MWC2015上发布。由于有Valve的SteamVR提供的技术支持，因此在Steam平台上已经可以体验利用Vive功能的虚拟现实游戏。



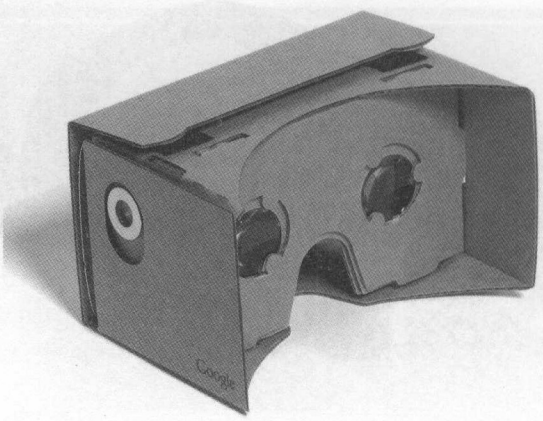


（3）PS VR 是索尼推出的 VR 头显，基于 PS4 主机的平台，玩家可以玩到丰富的 VR 内容。在 2016 年的 E3 发布会上，索尼也公布了包括《最终幻想 15》在内的一系列 PS VR 作品，满足了玩家的期待。



有不少人认为，与手机搭配的移动 VR 设备才是未来 VR 的发展方向，因此有很多厂商也发布了他们的移动端 VR 设备。

（1）Google Cardboard 最初是谷歌法国巴黎部门的两位工程师大卫·科兹（David Coz）和达米安·亨利（Damien Henry）的创意。他们利用谷歌“20% 时间”规定，花了 6 个月的时间打造出这个实验项目，意在将智能手机变成一个虚拟现实的原型设备。



（2）Gear VR 又名三星 Gear VR，是三星推出的一款 VR 头显，用户必须有一台配套的三星 Galaxy Note 4 手机，并确保它的系统升级到最新版本，最后用户必须把 Gear VR 附带的 16G microSD 卡插入手机中，因为这款产品需要 Note 4 作为显示屏才能提供相关的体验。



基于移动手机的 VR 设备还有很多，我们在此不一一列举，在本书，我们采用 Unity 3D 引擎讲解 VR 产品的开发。关于硬件平台，移动端 VR 设备我们主要选择基于 Google Cardboard 进行技术讲解，这也是充分考虑到谷歌纸盒的价格非常平民化，每个人都能够轻易获取，再加任意一台智能手机就可以进行开发和测试了。其他移动 VR 应用开发的原理与 Google Cardboard 类似。

另外，本书将重点讲解基于 HTC Vive 开发 VR 产品，详细说明 HTC Vive 中手柄事件的获取及处理，并附带开发案例，希望能给大家的学习和工作提供帮助。

本书最后还简单介绍了其他 VR 设备的一些知识作为参考。

第14章 基于 Google Cardboard 开发 VR 应用

14.1 Cardboard SDK for Unity 环境配置

本节将逐步介绍如何构建和部署 Cardboard SDK for Unity 中附带的演示场景。

1.SDK 和演示场景的导入和运行

（1）下载并安装 Unity（本书采用 Unity 5.3.4f1 版本）。

（2）通过以下链接下载 Cardboard SDK for Unity 软件包：

https://github.com/googlesamples ... itypackage?raw=true。

（3）通过以下链接下载 Cardboard for Unity 演示场景：

https://github.com/googlesamples ... itypackage?raw=true。

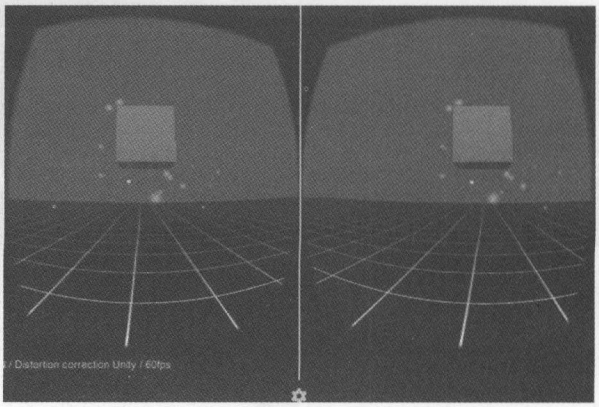
（4）打开 Unity，创建一个新项目，名称为 CardboardDemo。

（5）导入 SDK 软件包：Assets → ImportPackage → Custom Package。选择 Cardboard SDKForUnity. unitypackage 下载软件包并点击 Open。确保已勾选 ImportingPackage 对话框中的所有复选框，点击 Import。

注意：

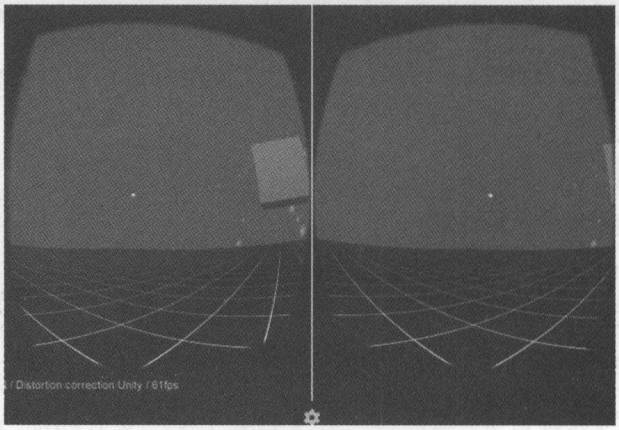
如果你使用的是 Unity 5，系统会警告 API 将自动升级。如果出现上述警告，请接受它并继续操作。

（6）导入演示软件包：Assets → Import Package → Custom Package。重复上述步骤，这次选择 CardboardDemoForUnity unity package。在编辑器的 Project 窗格中，导航至 Assets → Cardboard → DemoScene 文件夹，然后打开 DemoScene。你应该能够看到这样的一个场景，在贴有纹理的网格平面上漂浮着一个立方体。

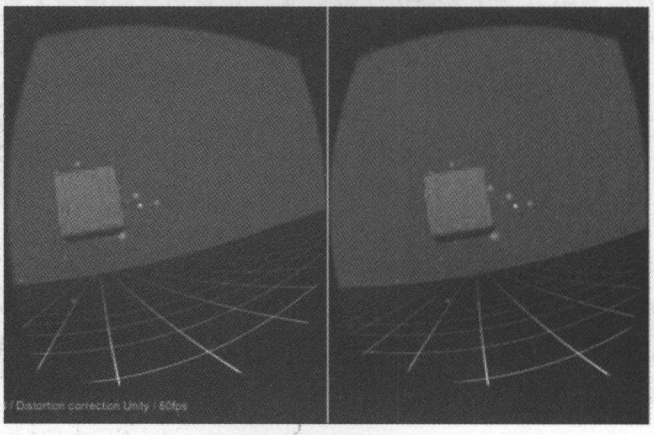


（7）按 Play，你会看到一个游戏视图，上面显示呈现立体感的红色立方体。在播放模式中，按 Alt 并移动鼠标可沿水平方向和垂直方向平移，从而模拟头部移动。使用 Ctrl 模拟头部从一边向另一边倾斜。

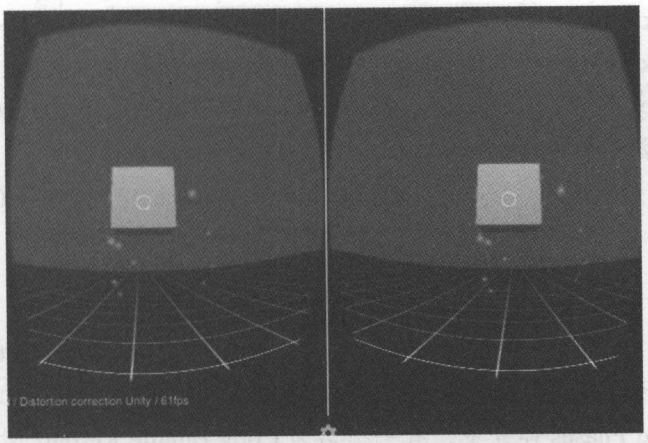
使用鼠标 +Alt 键模拟头部移动。



使用鼠标 +Ctrl 键模拟头部倾斜。



焦点选中后变大，并且被选中的立方体变成绿色。



2. iOS 平台的发布和设置

构建 Unity Project 的步骤如下：

（1）转到 File → Build Settings，打开 Build Settings。

（2）在 Platform 下选择 iOS，然后点击 Switch Platform 按钮。

（3）点击 PlayerSettings 打开 Inspector 中的 PlayerSettings 面板。

（4）在 PlayerSettings 中，在 iOS 设置的底部，点击 Resolution and Presentation 可展开该面板，然后将 Default Orientation 设为 Landscape Left。

（5）点击 Other Settings 以展开该面板，并查找 Bundle Identifier 字段。为应用输入有效的软件包名称（如 com.mycompany.cardboarddemo），该名称须遵循 Java 软件包和 .Net 命名空间的命名规则。该名称格式通常为 com.<yourcompany>.<appname>。

（6）点击 Build，系统会提示你保存名称，并在 [您的 Unity 项目]/[您的保存名称] 目录中准备一个 XCode 项目，我们将该项目部署到 iPhone。

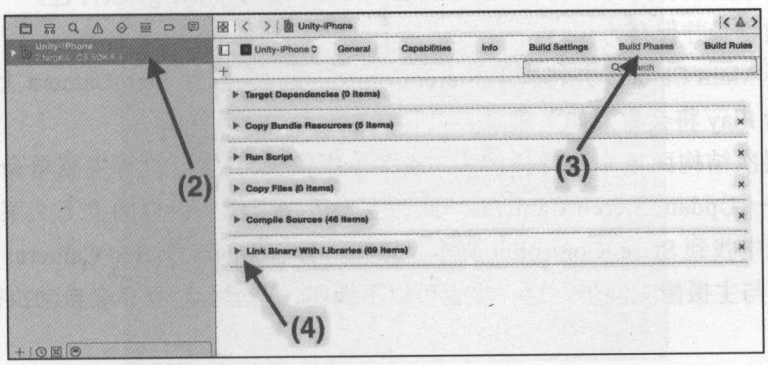
在 XCode 中部署项目的步骤如下：

（1）打开 XCode，从 File → Open... 菜单中打开你创建的 XCode 项目。它将成为你在上一个部分刚保存的目录中名为 Unity-iPhone.xcodeproj 的文件。

（2）点击 Unity-iPhone 项目并选中它。

（3）在项目菜单栏中，点击 Build Phases 以打开该部分。

（4）展开 Link Binary With Libraries 列表。



（5）点击 Link Binary With Libraries 列表最底部的“+”。这将打开一个窗口，上面显示“Choose frameworks and libraries to add”（选择要添加的框架和库）。

（6）对于 Unity 5.1.1 之前的版本，在搜索框中输入 CoreText，出现 CoreText.framework 时，双击它以将其添加到库中。

（7）如果一切进展顺利，你应该会看到“Build Succeeded”，并且 Cardboard 演示版将在 iPhone 上启动！

注意：

在 XCode 中，你可能会遇到一个构建错误，显示“Multiple build commands to output file [yourproject] resources.bundle”。如果看到此错误，表明你可能有第二个构建指令。在 Xcode 项目的 Build Phases 部分中，有一个名为 Copy Bundle Resources 的部分，resources.bundle 已出现两次，删除其中的一个。

3.Android 平台的发布和设置

（1）在 File → Build Settings 中选择 Android 作为平台。

（2）点击 Player Settings，在“Other Settings”下面将软件包名称输入“Bundle Identifier”字段（例如，com.example.CardboardUnityDemo）。在“Resolution and Presentation”下面检查默认方向是否设为“Landscape Left”。

（3）点击 Build and Run。如果系统提示输入 Android SDK 的位置，请选择已解压的 SDK。

（4）将导出的 apk 文件安装至 Android 系统手机中，打开应用，插入 Cardboard 即可。

14.2 利用 Cardboard 创建 VR 场景

本节将介绍如何使用 Cardboard SDK for Unity 中的脚本和预制件结合 Unity 3D 创建自己的 VR 场景。

1. 通过脚本创建 VR 场景

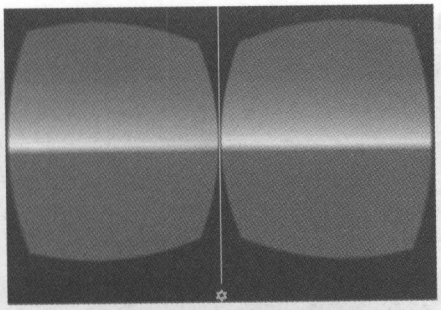
（1）创建新的项目并导入 CardboardSDKForUnity.unitypackage 软件包。

（2）新建场景并保存为 Test。

（3）选择 Main Camera 并将脚本 StereoController.cs 附加到 MainCamera 上。

（4）点击 Play 将会看到如下效果。

（5）在层次结构面板中选择摄像头（或多个摄像头），然后执行主菜单命令 Component → Cardboard → Update Stereo Cameras。此命令也可在摄像头组件的上下文菜单中找到：在 Inspector 面板中找到 StereoController 脚本，该脚本具有 UpdateStereo Cameras 按钮，点击可将立体摄像头与主摄像头同步。这一步也可以不操作，在运行时程序会自动执行。



2. 通过预制件创建 VR 场景

除了使用脚本创建 VR 以外，还可以使用 Cardboard 预制件（专为不适合采用上面步骤的情况而设计）添加 VR。

对于具有单目摄像头的全新项目，可考虑使用 CardboardMain 预制件替换主摄像头。此预制件包含完整的立体影像装置，以及标记为 MainCamera 的摄像头和一个用于控制 VR 模式的 Cardboard 脚本的实例。

对于已使用 StereoController 脚本填充的现有项目，添加一个 CardboardAdapter 预制件作为子项，这会将 CardboardHead 置于主摄像头下面，从而使用户的头部运动只影响到立体摄像头。

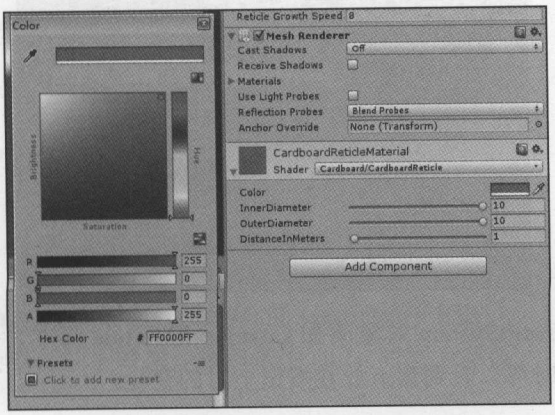
14.3 Cardboard 中 GUI 的设置

在开始讲 GUI 设置之前，先介绍一下 VR 场景中的焦点设置。

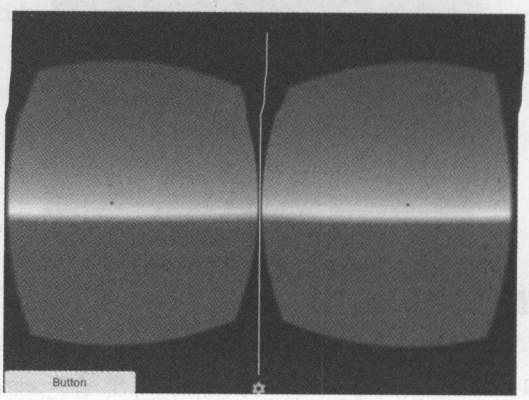
1.VR 场景中的焦点设置

（1）在 Cardboard → Prefabs → UI 目录下，将预制件 CardboardReticle 放置在 Stereo Controller 脚本所依赖的对象下面，作为 StereoController 所依赖游戏对象的子物体。

（2）选择新拖入的 CardboardReticle 对象，并将材质上面的 Color 属性设置为红色，点击 Play 即可看到下图的效果，也就是在 VR 场景中添加了跟随用户头部运动的焦点。



运行后位于屏幕中心的红色焦点如下所示。

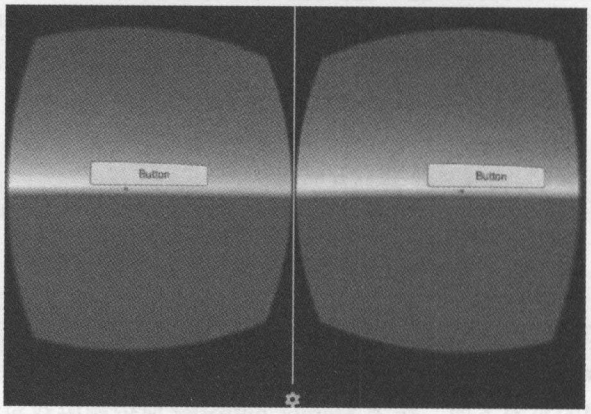


2. GUI 事件触发

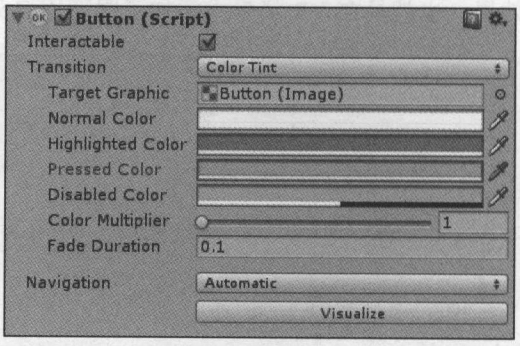
（1）创建 button 按钮，在 Unity 中通过 GameObject → UI → Button 创建一个 button 按钮。

（2）选择 Canvas，将 Render Mode 设置为 WordSpace，并调整 button 在屏幕中的位置。

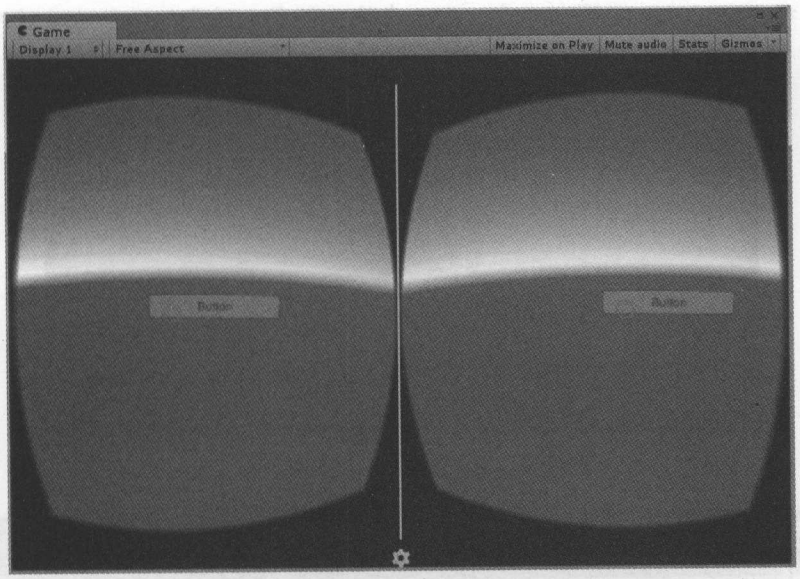
（3）点击 Play 后将会在 VR 模式下看到镜头中的 UI 按钮。



（4）设置 button 高亮并设置按下的颜色，以便较为清晰地观察聚焦和点击效果。



（5）给场景中的 EventSystem 对象添加 GazeInputModule 脚本，并删除其上的 Standard alone mudule 组件。此时可以看到焦点放在按钮上时的变化。



14.4 Cardboard 中 3D 物体的选取

（1）按照前面几节的内容创建 VR 场景和焦点。为了与场景中的 3D 对象交互，还需要将 PhysicsRaycaster 组件添加到 Main Camera。

（2）在场景中创建一个 Cube，添加材质并设置为白色，将 Cube 的位置设置到屏幕中合适的地方。

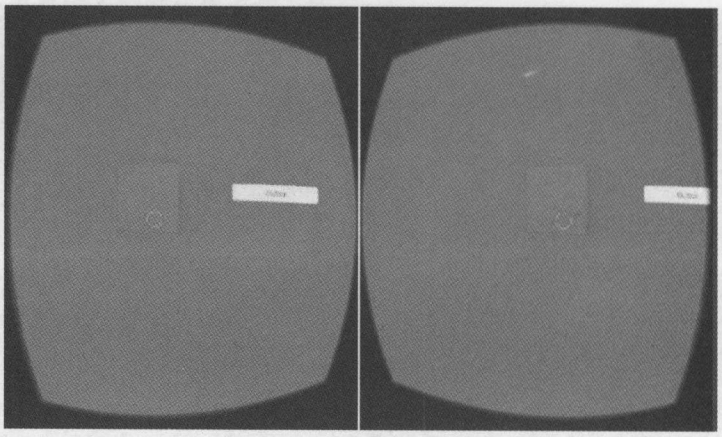
（3）为需要选取的 3D 对象 Cube 添加碰撞器组件。

（4）新建 Cube.cs 脚本，并添加如下代码。

```
public class Cube : MonoBehaviour
{
   public void GazeStart()
   {
   GetComponent<Renderer> ().material.color = Color.blue;
   }
   public void GazeExit()
{
        GetComponent<Renderer> ().material.color = Color.white;
   }
}
```

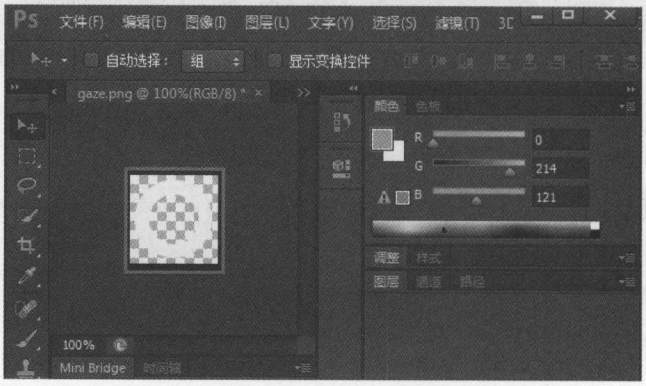
（5）为需要选取的 3D 对象 Cube 添加 EventTrigger 组件，并添加 PointerEnter 和 PointerExit 两种事件，分别调用 Cube.cs 中的 GazeStart() 和 GazeExit() 方法。

（6）点击 Play 则会看到如下效果，焦点选中 Cube 时变为蓝色。

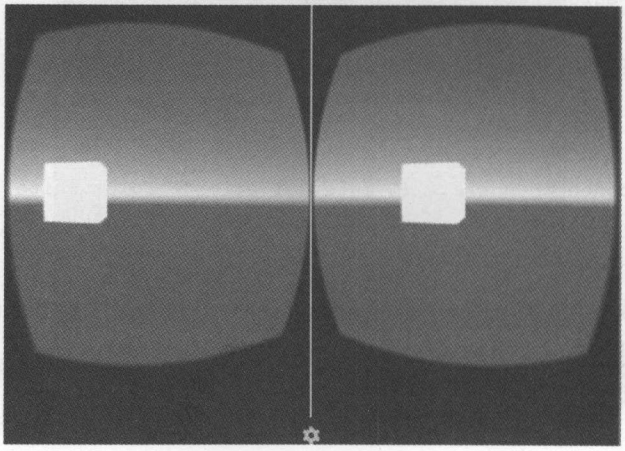


14.5 Cardboard 中凝视触发操作的实现

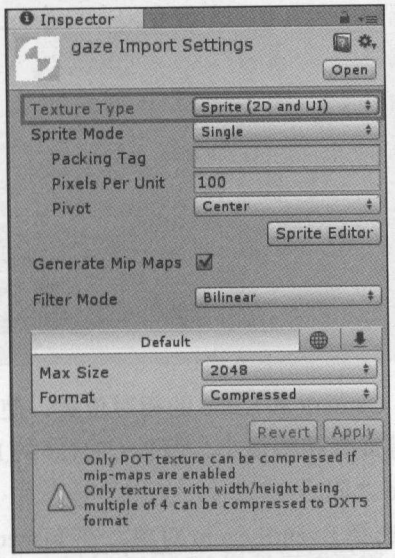
在开始本节内容前需要在 Photoshop 中制作如下图标。



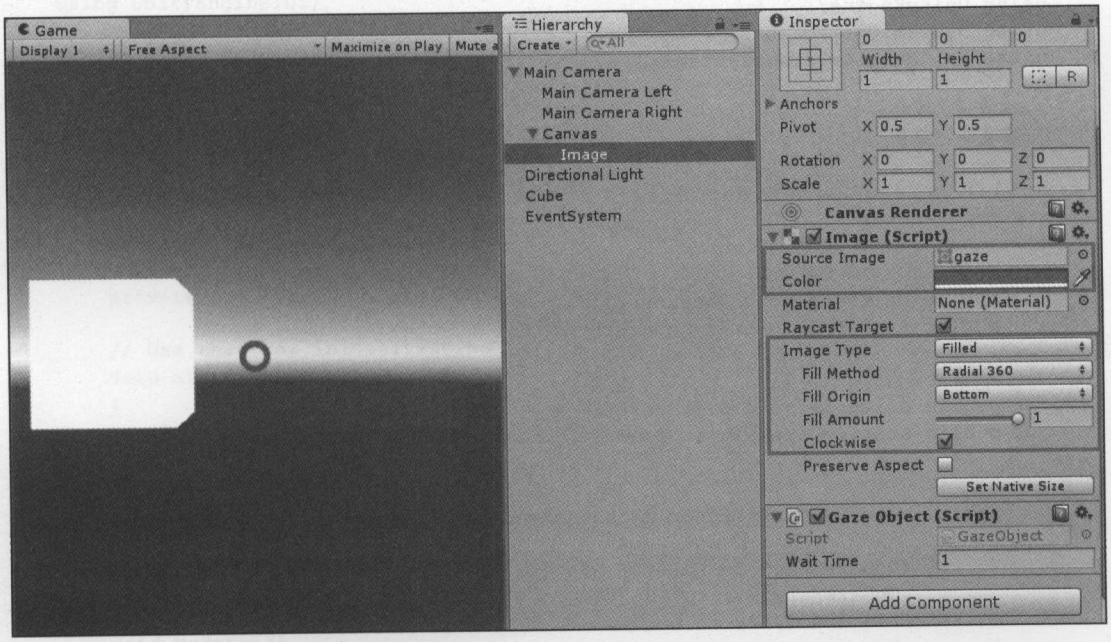
（1）根据前面几节的内容创建一个 VR 场景，并在该场景中创建一个 Cube。



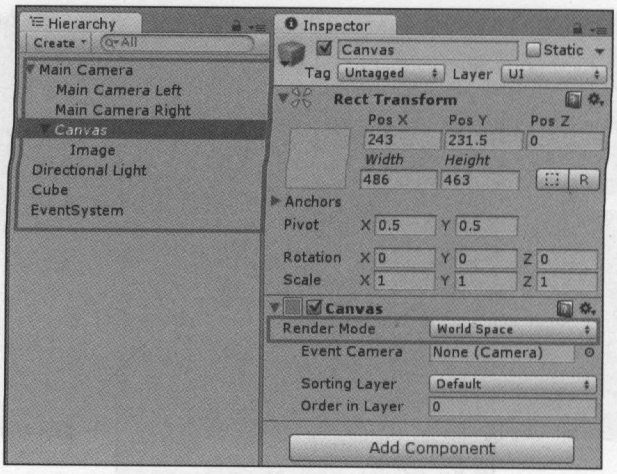
（2）导入预先制作好的图标，将其命名为 gaze。点击该图标，在 Inspector 面板下找到 Texture Type 属性，将其设置为 Sprite（2D and UI）。



（3）创建一个 Image，在 Unity 中选择 GameObject → UI → Image 就可创建一个 Image。在 Inspector 面板中的 Image 组件中找到 Source Image 属性，将上一步中处理的图片拖动到该属性对话框上，将 Image Type 设置为 Filled，调整 Iamge 到合适大小。



（4）选择 Canvas，在 Inspector 面板中移除 Canvas Scaler 和 Graphic Raycaster 两个组件，将 Canvas 组件的 Render Mode 设置为 World Space。在 Hierarchy 面板中将 Canvas 拖动到 Main Camera 上，使其作为 Main Camera 子对象。



（5）调整 Canvas 与 Main Camera 和 Cube 的相对位置，使得 Canvas 和 Main Camera 的位置坐标中 PosX 和 PosY 相等或者相差很小，调整 Canvas 的 PosZ 使图标 gaze 出现在场景中并处于 Main Camera 和 Cube 之间。

（6）在 Hierarchy 面板中选择 EventSystem 对象，在 Inspector 面板中移除其 Standalone Input Module 组件，在 Assets 面板下选择 Cardboard → Scripts → Gaze Input Module，将其拖到 EventSystem 对象上。

（7）编写脚本 GazeEventObject，代码如下：

```
using UnityEngine;
using System.Collections;
using UnityEngine.EventSystems;

//焦点进入事件
//焦点退出事件

public class GazeEventObject : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler
{
    //焦点进入的委托类型
    public delegate void GazeEnter();
    public static GazeEnter OnGazeEnter;

    //焦点退出的委托类型
    public delegate void GazeExit();
    public static GazeExit OnGazeExit;


    public void OnPointerEnter (PointerEventData eventData)
    {
        if (OnGazeEnter != null) {
            OnGazeEnter();
        }
    }

    public void OnPointerExit (PointerEventData eventData)
    {
        if (OnGazeExit != null)
        {
```

```
            OnGazeExit();
        }
    }

}
```

（8）编写脚本 CubeEvent，代码如下：

```
using UnityEngine;
using System.Collections;

public class CubeEvent : MonoBehaviour {

    void Start ()
    {
        GazeObject.OnWaitEvent += OnWaitOver;
    }

    void OnWaitOver()
    {
    transform.Rotate (0, 60, 0);
    }
}
```

（9）将脚本 GazeEventObject 和 CubeEvent 保存后挂载到 Cube 对象上。

（10）编写脚本 GazeObject，代码如下：

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class GazeObject : MonoBehaviour
{
    public delegate void OnWaitTrigger();
    public static OnWaitTrigger OnWaitEvent;

    public float WaitTime = 5;

    private bool start = false;

    private float timeGo = 0;

    // Use this for initialization
    void Start ()
    {
        GazeEventObject.OnGazeEnter += OnGazeEventEnter;
        GazeEventObject.OnGazeExit += OnGazeEventExit;

        start = false;
        timeGo = 0;
    }

    // Update is called once per frame
    void Update ()
    {
        if (start)
        {
            timeGo += Time.deltaTime;
```

```
            Image img = GetComponent<Image>();

            img.fillAmount = 1 - (timeGo / WaitTime);

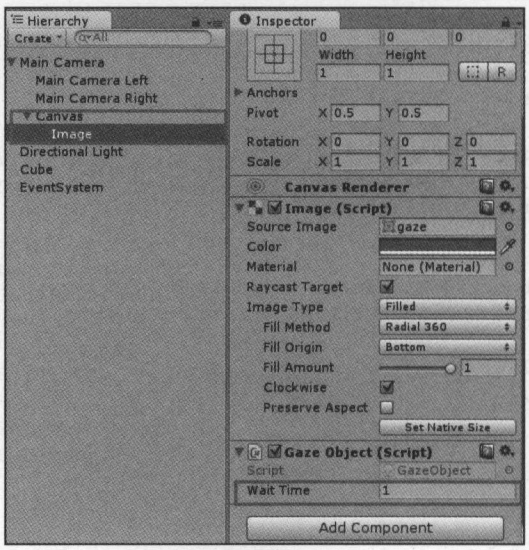
            if(timeGo > WaitTime)
            {
                start = false;

                if(OnWaitEvent != null)
                {
                    OnWaitEvent();
                    timeGo = 0;
                }
            }
        }
    }

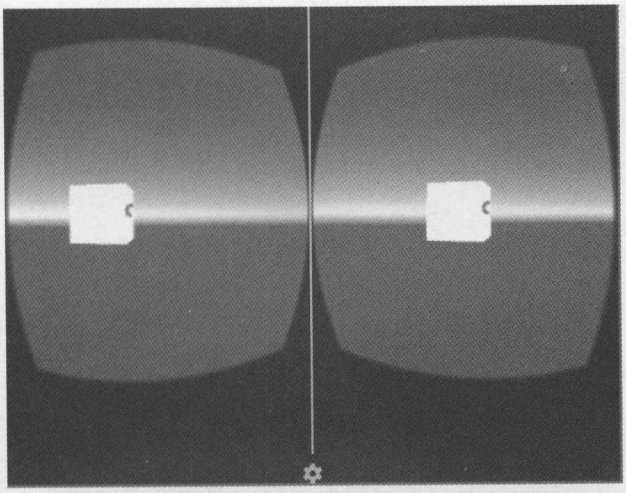

    void OnGazeEventEnter()
    {
        start = true;
    }

    void OnGazeEventExit()
    {
        Image img = GetComponent<Image>();
        img.fillAmount = 1;
        start = false;
    }
}
```

（11）将脚本 GazeObject 挂载到 Main Camera → Canvas → Image 对象上，将 Wait Time 设置为 1。



（12）点击运行，按下 Alt 键移动鼠标使得 gaze（红色图标）移动到立方体上，等待 1s 后立方体旋转，证明凝视触发操作已经实现。

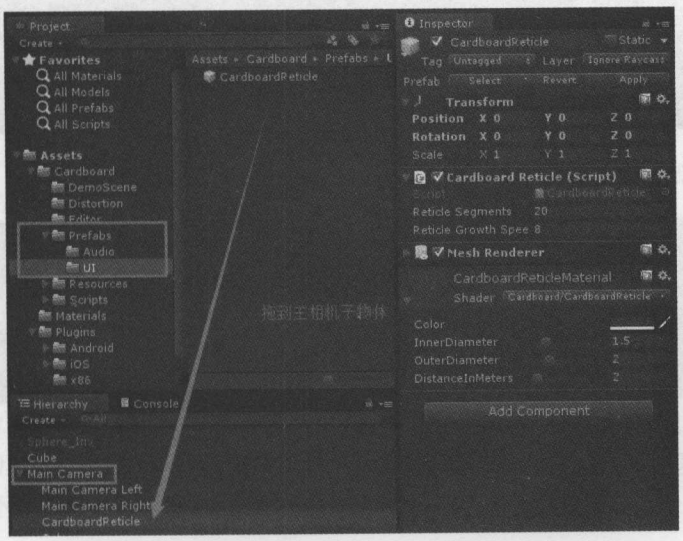


14.6 Cardboard 中全景相册的实现

这一节我们使用谷歌 Cardboard 制作一个全景相册，首先确保准备好以下工具：

- Cardboard 眼镜。
- 5.1 以上版本的 Unity 编辑器（本书使用 5.3.4f1 ）。
- 5 ～ 8 张全景图片。
- 使用 Max 等三维软件制作的一个法线向内的球。

（1）打开 Unity，新建一个场景，导入 Cardboard SDK for Unity，在当前场景中的 Main Camera 上添加脚本 StereoController，点击 StereoController 下的 Update Stereo Camera，按照下图进行操作。

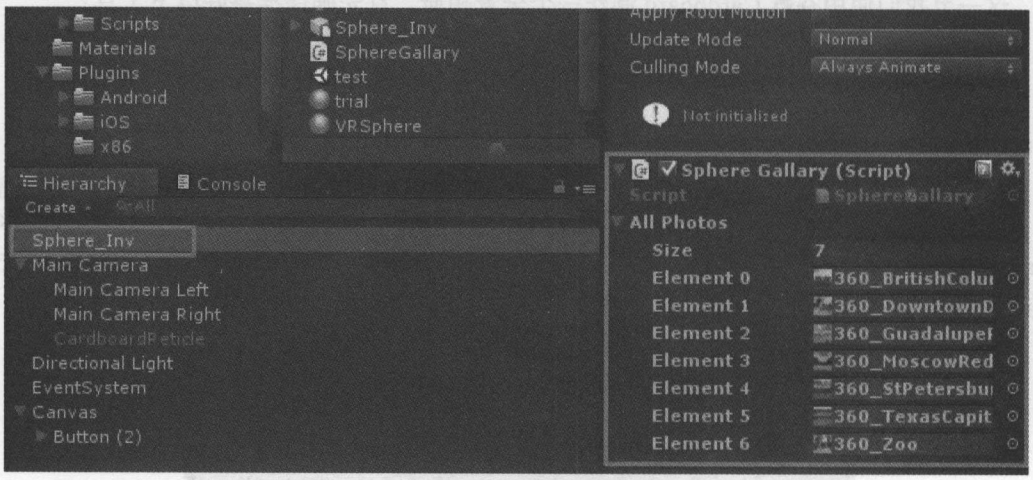


（2）将制作好的法线向内的球拖到场景中，同时创建一个脚本，命名为 SphereGallary，在脚本中编辑如如下代码并保存。

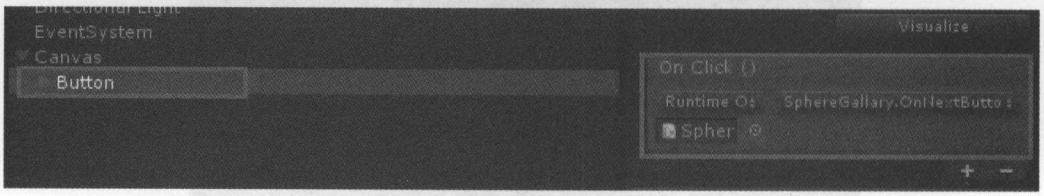
```
using UnityEngine;
using System.Collections;

public class SpherGallery : MonoBehaviour {
    public Texture[] AllPhotos;
    int currentIndex = 0;
    public void OnNextButtonClick()
    {
        Material mat = GetComponent<MeshRenderer> ().material;
        currentIndex++;
        if(currentIndex >= AllPhotos.Length)
        {
            currentIndex = 0;
        }
        mat.mainTexture = AllPhotos [currentIndex];
    }
}
```

（3）将收集到的全景图片拖到法线球的如下位置。



（4）接下来创建一个按钮（button），然后在调用处连接这个脚本。



（5）在连接了按钮之后，点击运行进行测试，效果如下图所示。

（6）导出到手机，然后装到 Cardboard 中以进行观看。



14.7 Cardboard 中全景视频播放的实现

Unity 中的视频播放插件有很多，常用的有 MobileMovieTexture、VideoPlayback 和 EasyMovie Texture。前两者可以再 PC 上运行，但是需要安装 QuickTime，而 EasyMovieTexture 不支持 PC 运行，可以在移动端运行，无需安装 QuickTime。Unity 默认支持的视频格式分别有 .mov、.mpg、.mpeg、.mp4、.avi 和 .asf。一般我们在项目应用中采用 .mp4 文件进行视频的播放。在本节中，我们将介绍如何使用 Easy Movie Texture 来实现在 Andorid 手机上播放 .mp4 格式的全景视频。

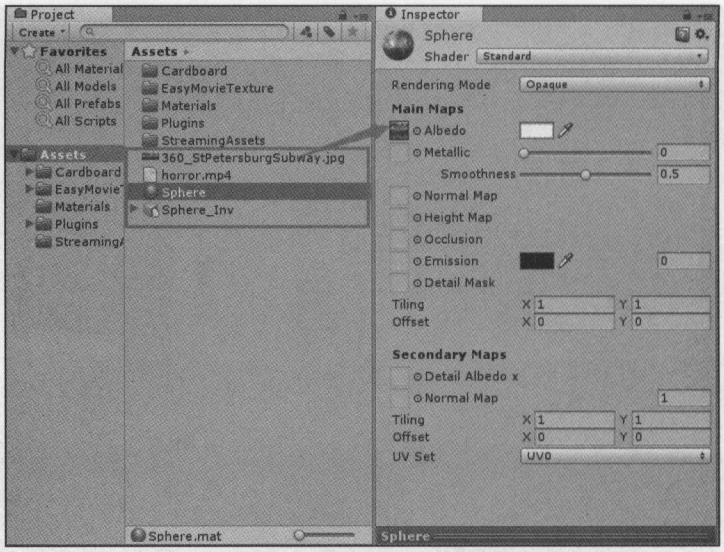
在开始学习前，需要准备以下工具：

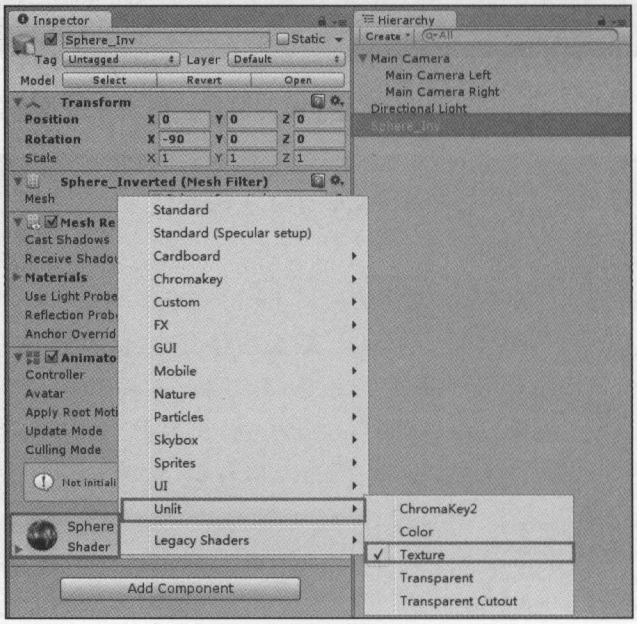
- Cardboard 眼镜。
- 5.1 以上版本的 Unity 编辑器（本书使用 5.3.4f1）。
- 一张全景图片和一个 .mp4 格式的全景视频。
- 使用 max 等三维软件制作的一个法线向内的球。
- Unity 视频播放插件 EasyMovieTexture。

（1）根据前面几节内容构建 VR 场景。

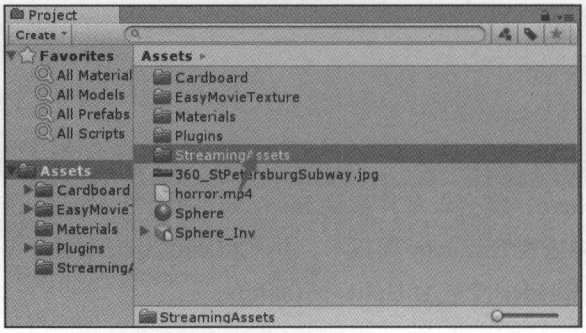
（2）导入插件 EasyMovieTexture、制作好的法线向内的球和全景图片。为导入的球体制作材质球，命名为 Sphere，将全景图片拖到 Inspector 面板中 Albedo 前面的方框内。

（3）将导入的球体拖到 Hierarchy 面板中，并把材质球 Sphere 赋给导入的球体，在 Inspector 面板中将 Sphere 下的 Shader 设置为 Unlit → Texture。

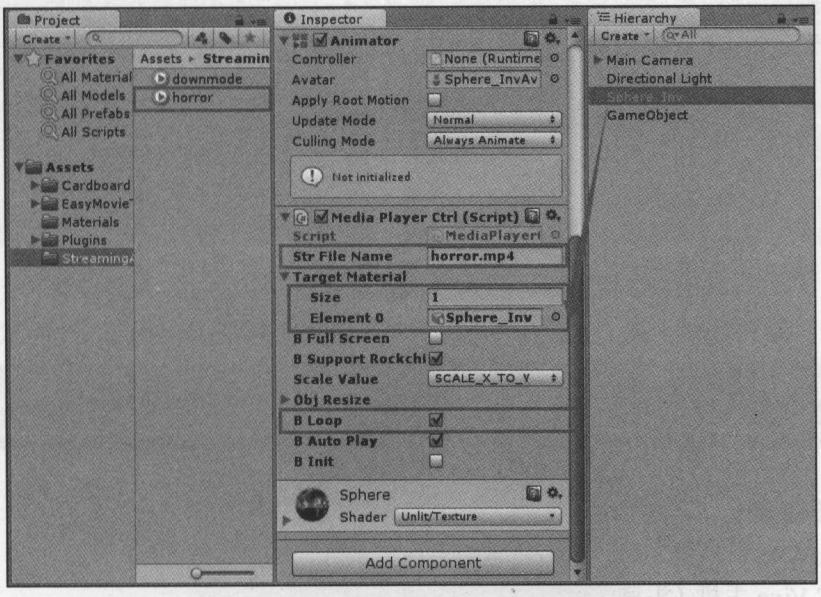




（4）在 Assets 面板中将全景视频拖动到 StreamingAssets 文件中。



（5）在 Assets 面板中按照路径 EasyMovieTexture → Scripts 找到脚本 MediaPlayerCtrl 并将该脚本挂载到导入的球体上。在 Hierarchy 面板中选中导入的球体，在 Inspector 面板中找到 MediaPlayerCtrl 组件，在 StrFileName 中手动填入“视频名 .mp4”（本例为 horror.mp4），其余选项按下图设置。



（6）为了能在视频播放完毕后在 Andriod 平台上按下返回键使得程序退出，需要创建脚本，命名为 APP，代码如下：

```
using UnityEngine;
using System.Collections;

public class APP : MonoBehaviour {

    void Update () {
    if (Input.GetKeyDown (KeyCode.Escape)) {
            Application.Quit();
        }
    }
}
```

（7）保存场景，将该场景导出成 apk，安装到 Android 手机上，点击运行即可观看。

第15章 基于 HTC Vive 开发 VR 应用

15.1 Vive 设备的安装

1. 设备检查

在收到 Vive 正式版之后，先检查包装盒内的部件。

- HTC Vive 主机（头显）×1
- 转接盒 ×1
- HDMI 数据线 ×1
- USB 转换线 ×1
- Lighthouse 基站 ×2
- SteamVR 无线动作控制器 ×2
- 控制器充电器 ×2
- USB 充电线 ×2
- 12V Lighthouse 基站电源线 ×2
- 12V 接线盒电源线 ×1
- Narrow Face 头显垫圈 ×1
- Wide Face 头显垫圈 ×1
- Lighthouse 基站固定架 ×2
- HTC 耳机 ×1
- 使用手册 ×1

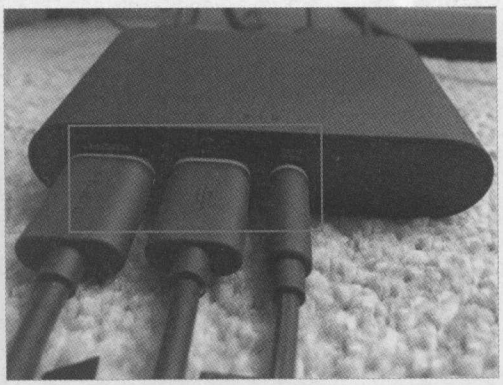
以上就是 HTC Vive 包装盒内的全套部件。

2. 设备安装

（1）将转接盒和你的 PC 通过 3.0 USB 接口连接，同时连接 HDMI 数据线，转接盒电源线接电源。

（2）将头显三根数据线分别对应下图接口连接。





当接通上述数据线之后，如果头显左侧显示灯变红，就说明连接正确。

（3）固定两个 Lighthouse 基站，原装设备需要固定在墙上，相对比较麻烦，也可以配一对摄像机专用支架来进行固定，基站尽量超过地面 2m 以上，且向下倾斜 30° ～ 50° 为宜。



连接好基站电源线之后，两个基站会显示绿灯，并且会显示当前该基站所处的频道。

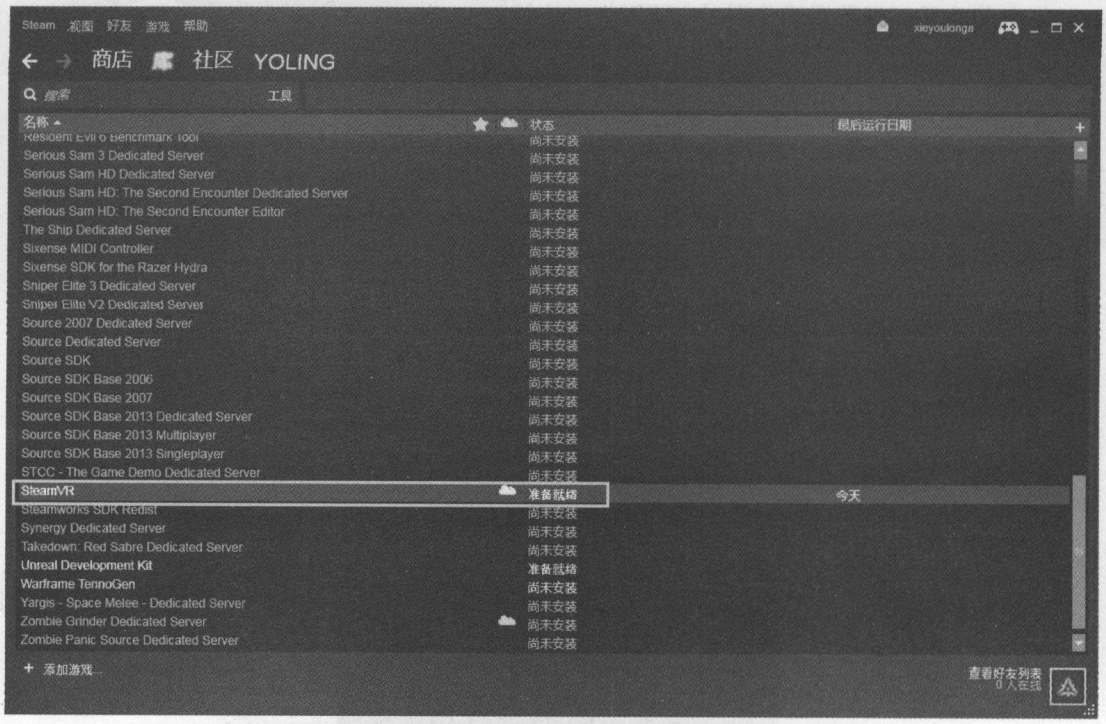
（4）打开两个动作控制器，把它和头显放在两个基站所放区域内。

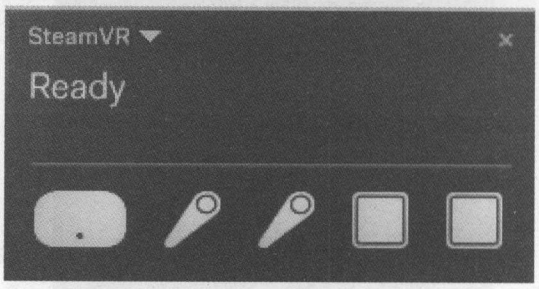
检查上述硬件设备是否连接好，查看各设备是否亮绿灯，如果全都没问题，下面进行软件平台搭建。

3. 软件调试

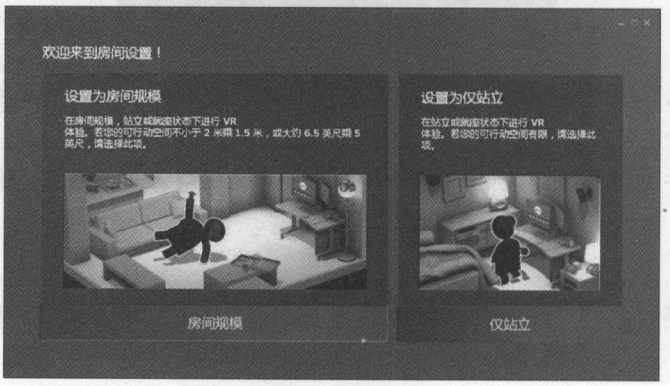
下载 Steam 平台，在 Steam 平台的工具库里找到 SteamVR 组件并下载安装，安装好之后点击运行。

在启动 SteamVR 之后，它会自动检测当前所连接的设备，如果所有设备都正常连接，则显示如下图。





点击 SteamVR 下拉菜单，选择房间设置：

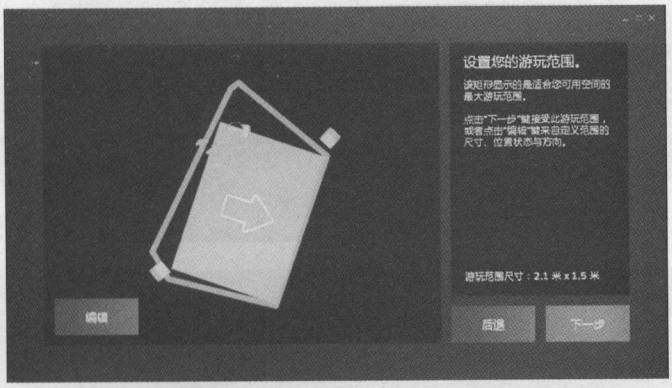


如果当前环境有足够的空间，则选择房间模式进行设置；如果空间太狭窄，则选择站立模式（这里选择房间模式进行调试）。





在这些步骤都完成之后，最后一步是进行范围划定，Vive 最远范围要求不超过 5m。



当这一步完成之后，就可以进行 Vive 体验了。

15.2 搭建 Vive 开发环境

在正式开始前，先确保配备有以下硬件设备：

- HTC Vive 完整设备一台。

- 操作系统：Windows 7 SP1，Windows 8.1，Windows 10。
- 处理器：Intel i5-4590，AMD FX 8350 或以上。
- 图形处理器：NVIDIA GeForce GTX 970，AMD Radeon R9 290 或以上。

如果以上硬件设备都满足了，那么开始我们的下面的旅程。

1.Vive 运行平台下载

HTC Vive 是由 HTC 与 Valve 共同开发的一款虚拟现实头盔产品。其中，由于 Valve 的 SteamVR 提供技术支持，所以在 Steam 平台上可以体验利用 VIVE 功能的虚拟现实游戏。

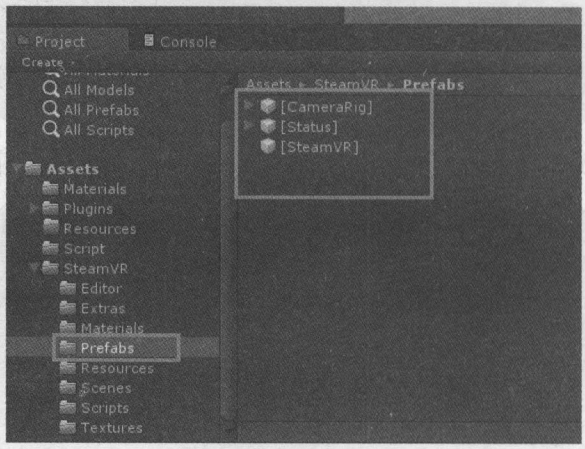
这里先下载 Steam 平台：http://store.steampowered.com/about/。

在下载安装好 Steam 平台之后，开始下载 SteamVR，在“库”→“工具”里面找到 SteamVR 并下载。

当 SteamVR 下载好之后，Vive 的运行环境就算是配置好了。

2. 下载用于 Unity 的 Vive 插件

创建一个新的工程，然后打开自带的资源商店，搜索 SteamVR Plugin 并下载导入，在导入之后，将所有默认的 GameObject 都删除，然后找到如下文件夹中的 Perfab，全部拖到场景中。



在做好这些准备工作之后，就可以戴上头盔然后点击运行进行测试了。

另外，大家可以参考 SteamVR Plugin 自带的示例场景，分别在下面这些路径下：

- SteamVR/Scenes/example
- SteamVR/Extras/SteamVR_TestIK
- SteamVR/Extras/SteamVR_TestThrow

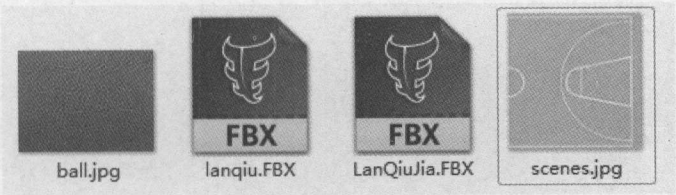
15.3 创建 Vive 游戏场景

首先打开 SteamVR 提供的示例场景 SteamVR_TestThrow，在该场景中，当你扣动

Trigger（手柄扳机键）时就会生成一个球体模型，然后抛出去。

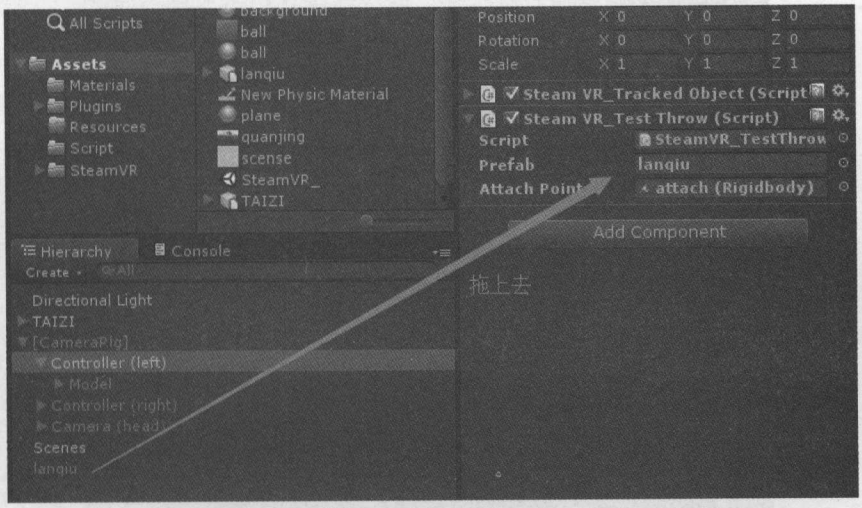
我们通过这个示例场景来制作一个简单的投篮游戏。

首先找一些资源，比如篮球架、篮球、地板贴图等，然后导入。

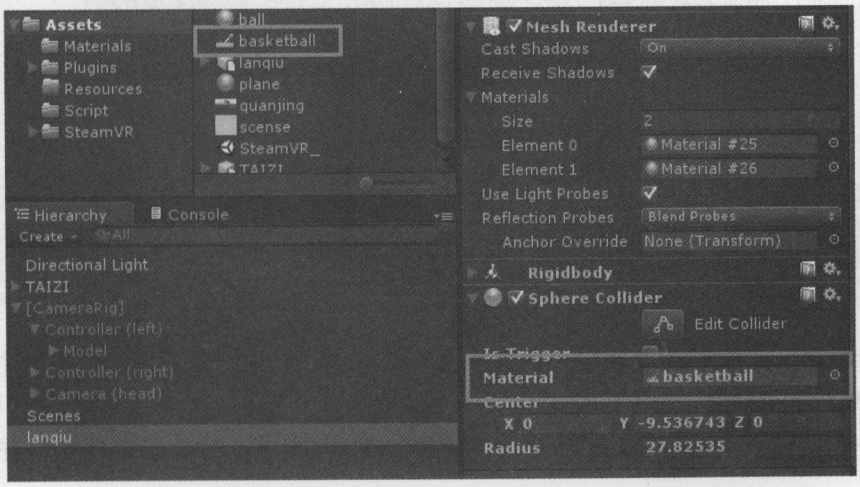


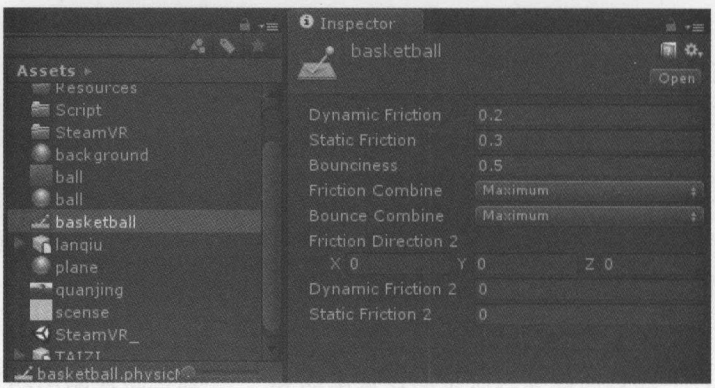
创建一个 Plane 的材质球并赋给当前场景中的 Plane，然后将资源中的 Scenes 贴图贴上去，并调整到合适位置。

接下来把做好的篮球架拖到场景中，并放到合适的位置。把场景中的 Template 删除，然后拖入篮球，并做如下改动。



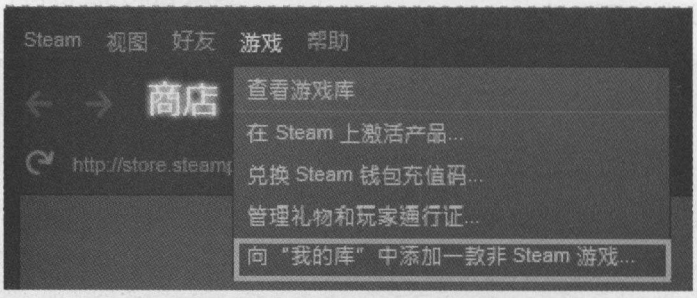
最后，给篮球创建一个物理材质，命名为 basketball，然后拖到篮球上。

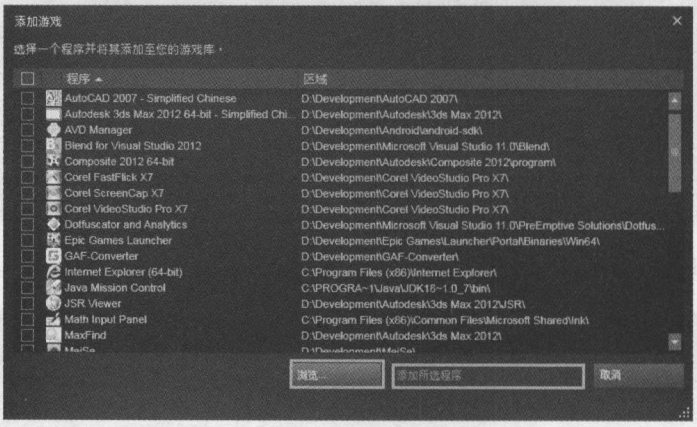




做完这一步，调整好设备，点击运行以测试效果。

最后，导出为 .exe 文件，然后加载到 SteamVR 平台。



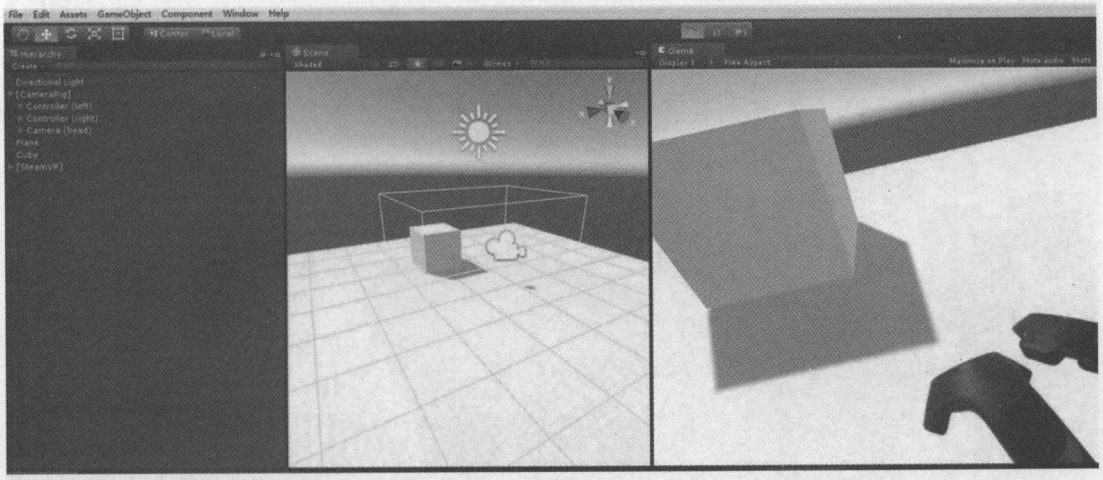


15.4 Vive 中的手柄交互

在 Vive 游戏中我们经常会用手柄去触碰一些东西，然后和它交互，那么这个功能是怎么实现的呢？这一节就为大家讲解 Vive 手柄触碰的开发。

新建一个 Unity 工程，导入 SteamVR 插件，找到 [CameraRig] 这个预制件并拖到场景

中，同时把场景中默认的照相机删除，然后再创建一个 Plane 和一个 Cube，点击运行进行观察。



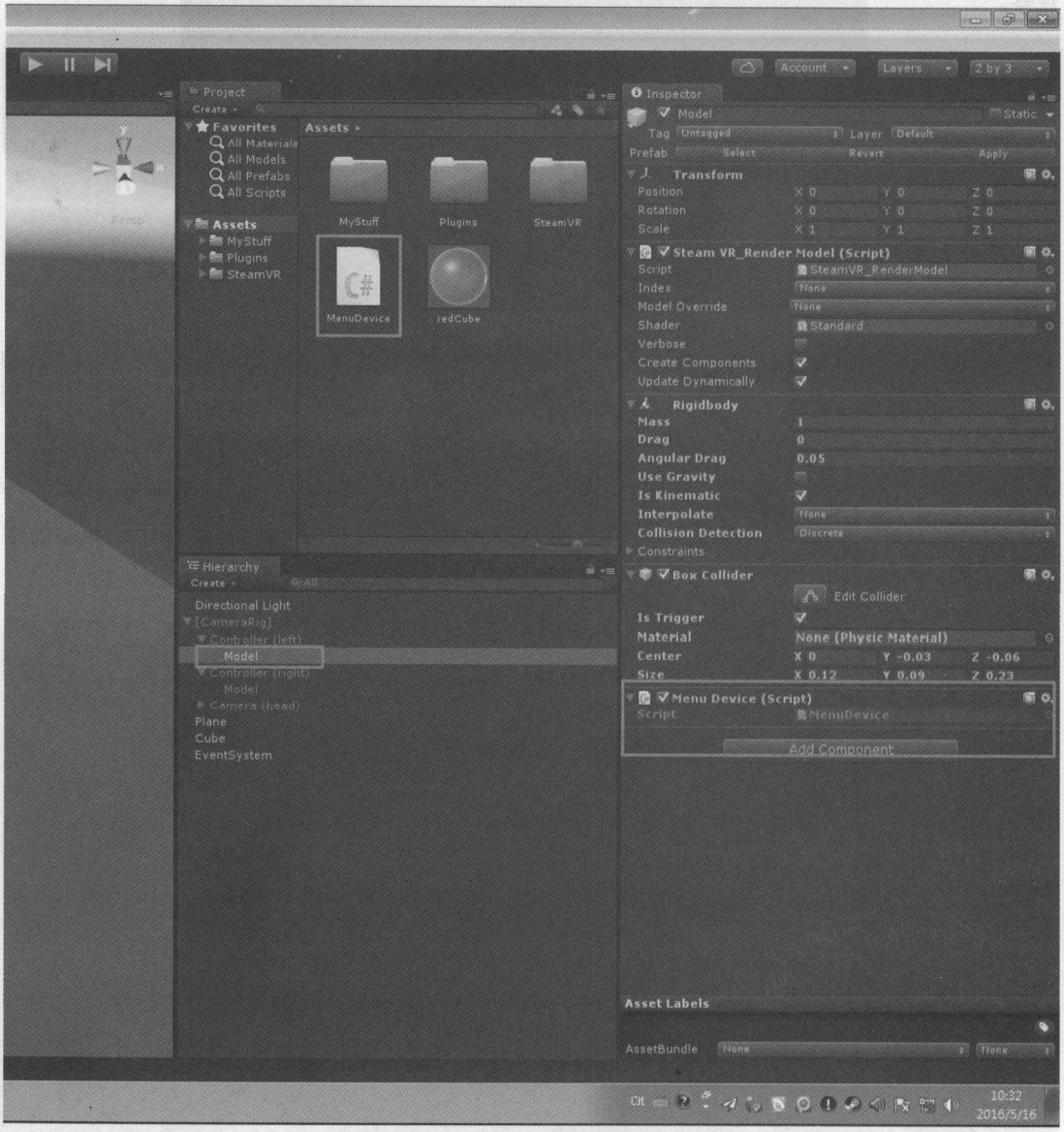
在 Scene 界面中选中其中一个动作控制器（下面简称手柄），添加一个碰撞器，并添加刚体，把碰撞器调整到合适大小。



创建 C# 脚本，改名为 MenuDevice，并写入如下代码：

```
public class MenuDevice : MonoBehaviour {
 void OnTriggerStay(Collider other)
    {
print(1231);
 }
}
```

将脚本拖到下图所示位置。

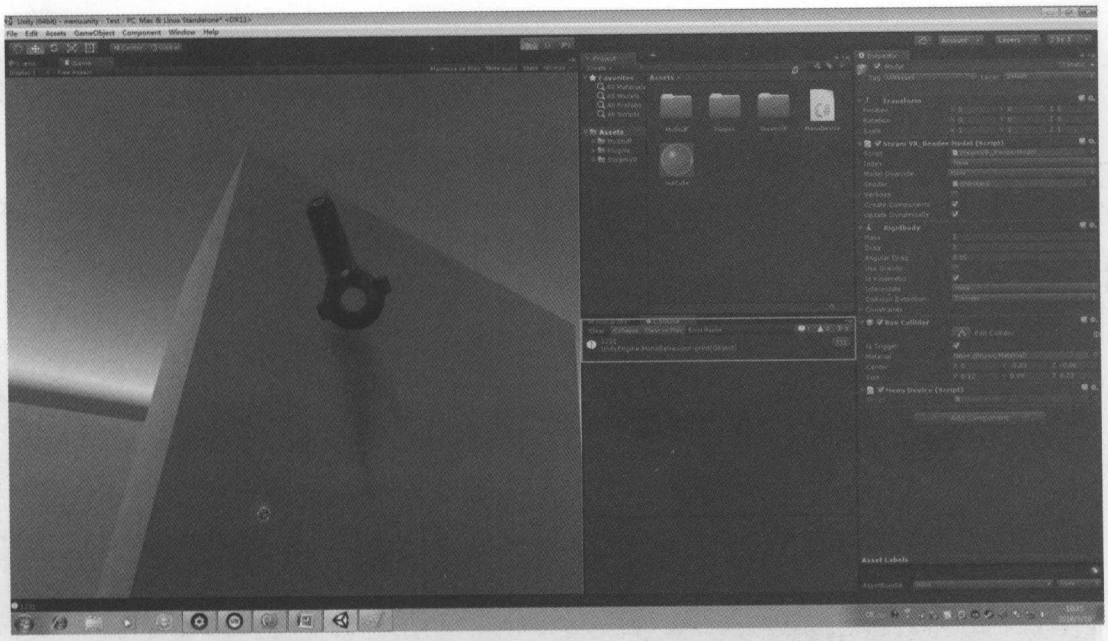


做完上一步之后，点击运行进行测试，把手柄靠近 Cube，然后看输出打印。

可以看到手柄碰到了 Cube 并且输出了我们所规定的内容。

接下来我们实现当手柄靠近并且扣动手柄扳机键（Trigger）的时候 Cube 会被拿起来的功能。

打开前面创建的 MenuDevice 脚本，写入如下代码：

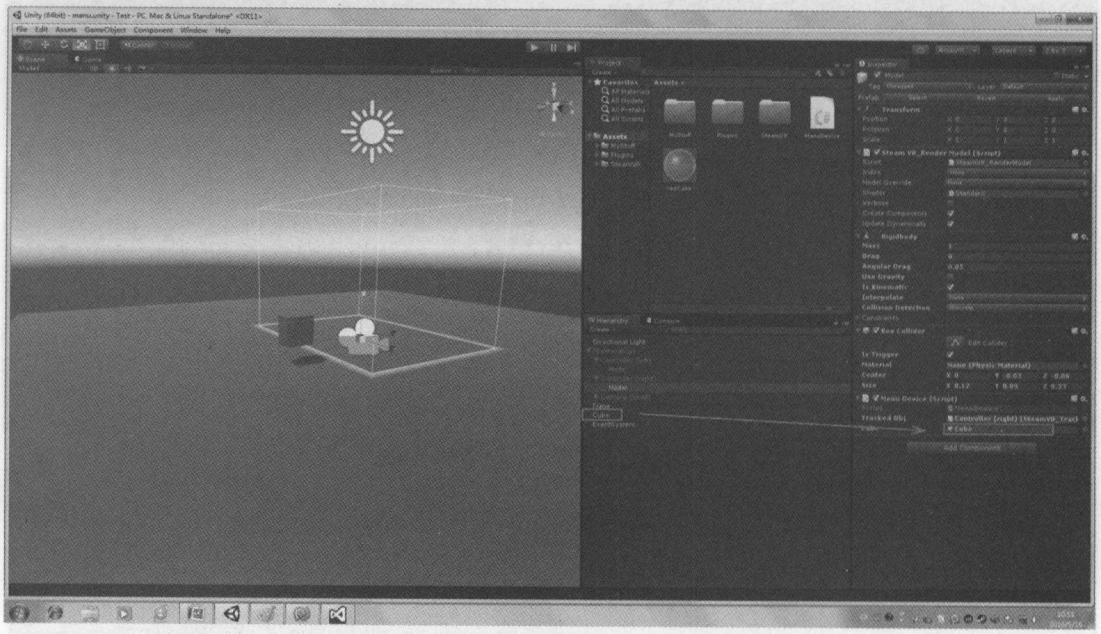


```
using UnityEngine;
using System.Collections;

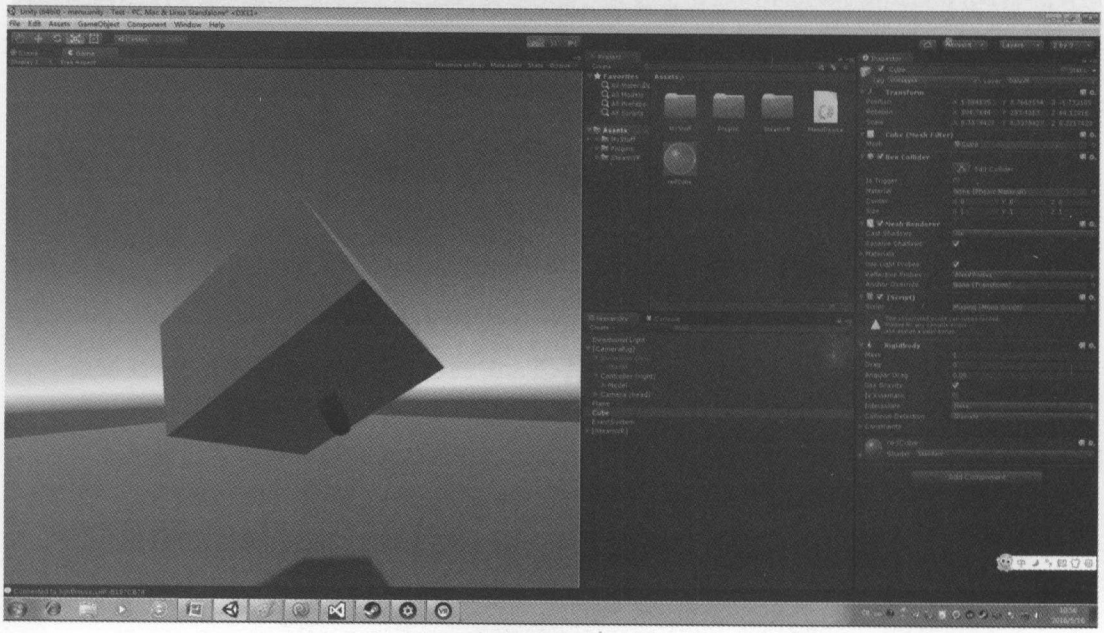
public class MenuDevice : MonoBehaviour
{
    public SteamVR_TrackedObject trackedObj;
    public GameObject Cube;
    bool isDrag=false;
    void Start ()
{
            trackObject =this.GetComponent<SteamVR_TrackedObject>();
    }
    void Update ()
    {
        if (isDrag == true)
        {
            Cube.transform.position = this.transform.position;
            Cube.transform.rotation = this.transform.rotation;
        }
    }
    void OnTriggerStay(Collider other)
    {
        //获得设备
           SteamVR_Controller.Device device = SteamVR_Controller.Input((int)
trackedObj.index);
        //按键按下
        if (device.GetTouch (SteamVR_Controller.ButtonMask.Trigger) == true) {
            isDrag = true;
        }
        else
        {
```

```
                isDrag = false;
        }
        }
}
```

在写完上述代码之后，在 Inspector 面板中将 Cube 拖到相关位置。



在做完之后点击运行进行测试，最后效果如下。

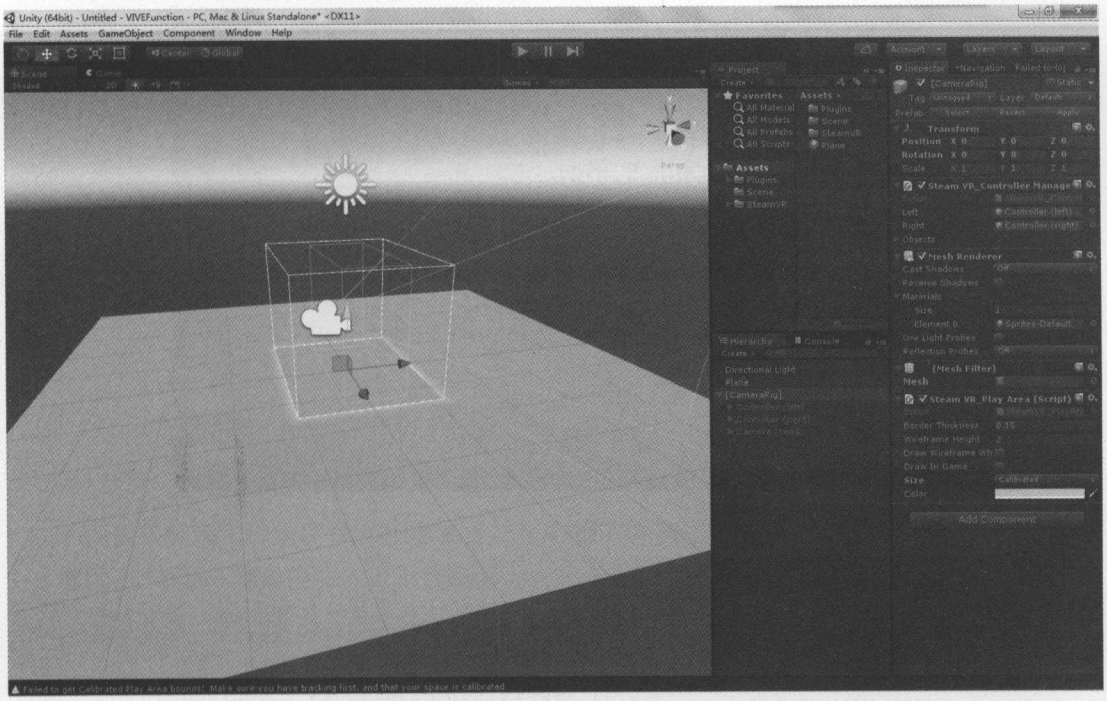


15.5 功能开发之扳机键（Trigger）

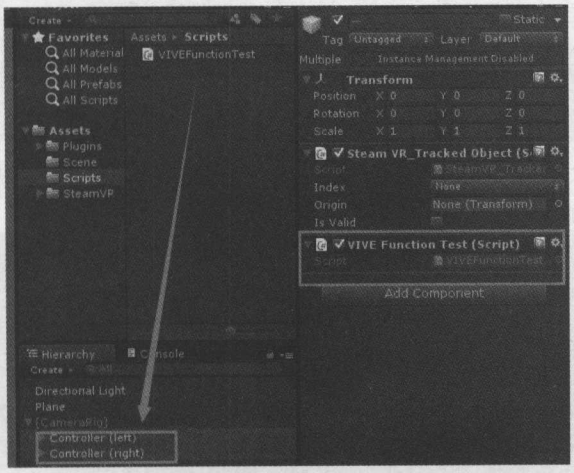
Vive 扳机键在 Vive 体验中起着非常关键的作用，大部分交互工作都是通过扳机键来触发的。

（1）下载 SteamVR 插件，创建新 Unity 工程并导入。

（2）搭建好 SteamVR 开发平台。



（3）创建新 C# 脚本，起名为 VIVEFunctionTest，把该脚本分别拖到左右摄像头上，并打开进行编辑。



（4）在进行手柄 Trigger 测试之前要先跟踪到手柄。

```
public class VIVEFunctionTest : MonoBehaviour
{

        SteamVR_TrackedObject trackObject;

        // Use this for initialization
        void Start ()
    {
            trackObject =this.GetComponent<SteamVR_TrackedObject>();
    }
.....
}
```

接下来获取具体扣动扳机的手柄，两个手柄在运行的时候，其 index 是不一样的，这里就是通过找到不同的 index 来找到相应的手柄。

```
void Update ()
    {
        SteamVR_Controller.Device device = SteamVR_Controller. Input ((int)trackObject.
index);
    }
```

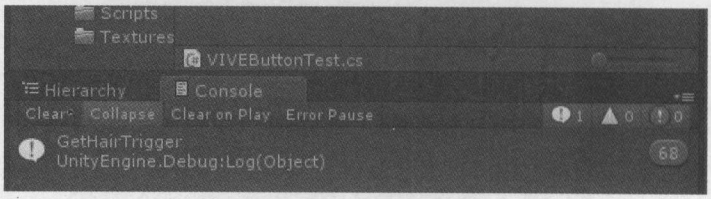
在获取到具体的手柄以后开始测试手柄扳机键的调用。

继续在 void Update() 函数中写入一下代码，并点击运行进行测试。

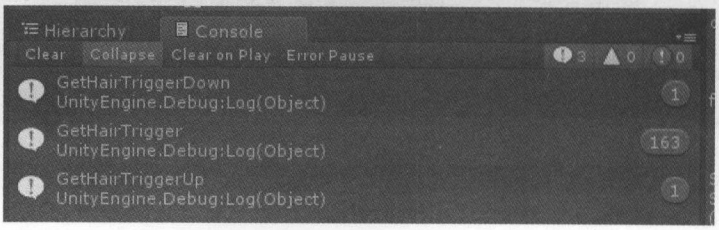
```
if (device.GetHairTrigger ())
{

    Debug.Log ("GetHairTrigger");
}
```

此时轻轻扣动扳机，便会在控制台看到输出的以下内容。



与 GetHairTrigger 同类型的还有两种形式，分别是 GetHairTriggerDown 和 GetHariTriggerUp，不同于 GetHariTrigger 的持续刷新，这两个函数在扣动扳机时候只会调用一次，将这三种函数结合起来会看到控制台输出如下内容。

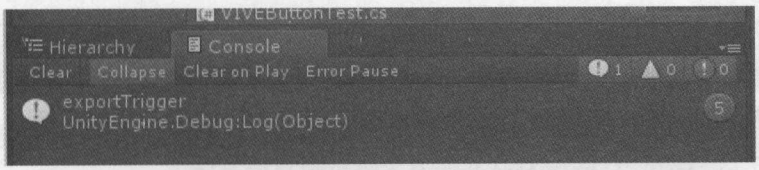


同时，对于调用手柄扳机键还有第二种方式，叫作 TouchDown()。

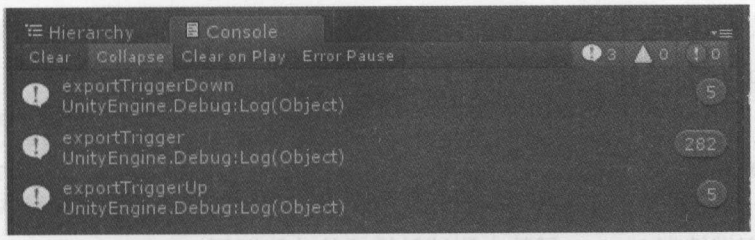
注释上述代码，继续在 Update 函数中写入以下代码。

```
if(device.GetTouchDown(Valve.VR.EVRButtonId.k_EButton_Axis1))
{
        Debug.Log ("exportTrigger");
}
```

不同于上述形式，该类型在用于手柄扳机键时要检测一次完整的输入，即按下、按住、抬起。



同样，GetTouchDown 也有三种不同的形式，这里还是直接看输出。

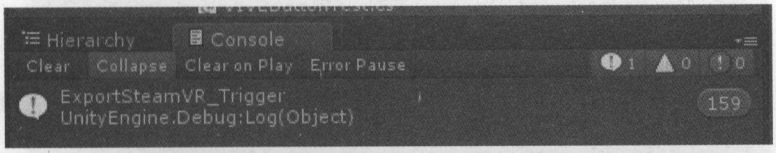


手柄扳机键除了以上两种调用形式外，还有第三种调用方式。

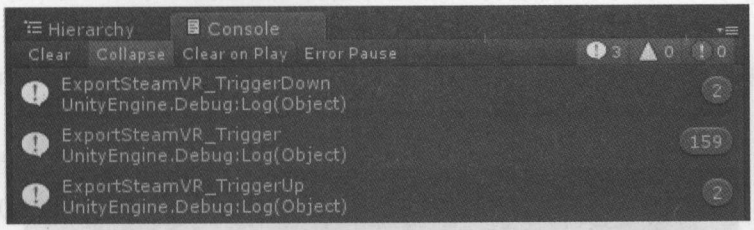
继续在 Update 函数中写入以下代码。

```
if(device.GetPress(Valve.VR.EVRButtonId.k_EButton_SteamVR_Trigger))
{
        Debug.Log ("ExportSteamVR_Trigger");
}
```

保存后点击运行，会有如下内容输出。



GetPress() 也有三种形式，这里我们直接输出控制台。



以上就是 HTC Vive 手柄中扳机键调用的三种方式，下面总结一下扳机相应事件：

- GetHairTrigger()
- GetTouch(Valve.VR.EVRButtonId.k_EButton_Axis1)
- GetPress(Valve.VR.EVRButtonId.k_EButton_SteamVR_Trigger)

15.6 功能开发之触摸板（Touchpad）

Vive 手柄触摸板即手柄正上方圆盘，它支持两个功能，滑动和按下。

1. 按下功能

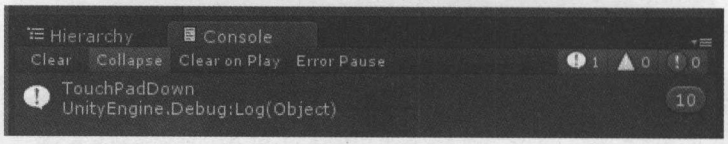
类似于扳机键的调用，在脚本开始前先跟踪调用到具体发射指令的手柄。

```
SteamVR_TrackedObject trackObject;
    void Start ()
    {
        trackObject =this.GetComponent<SteamVR_TrackedObject>();
    }
    void Update ()
    {
        SteamVR_Controller.Device device = SteamVR_Controller.Input ((int)
trackObject.index);
    }
```

接下来我们继续在 Update 函数中测试触摸板的功能调用。

```
if(device.GetTouchDown(Valve.VR.EVRButtonId.k_EButton_SteamVR_Touchpad))
{
    Debug.Log ("TouchPadDown");
}
```

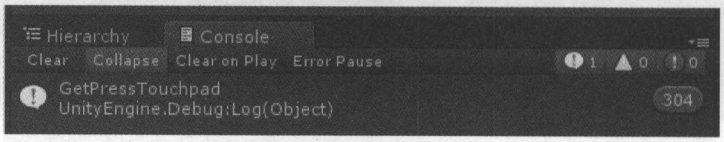
继续写入如上代码，然后点击运行。



接下来继续使用 GetPress() 来实现。在 Update() 函数中写入如下代码并检测。

```
if (device.GetPress (Valve.VR.EVRButtonId.k_EButton_SteamVR_Touchpad))
{
    Debug.Log ("GetPressTouchpad");
}
```

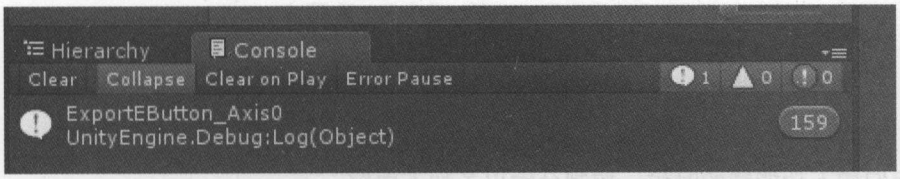
控制台输出内容如下。



触摸板按下也可以用下面的方法来调用：

```
if (device.GetPress (Valve.VR.EVRButtonId.k_EButton_Axis0))
{
        Debug.Log ("ExportEButton_Axis0");
}
```

控制台输出如下内容。



以上就是 Vive 手柄圆盘的按下功能调用，这里总结一下：GetTouch() 是在运行时轻轻点击即可触发，而 GetPress() 则需要触摸板完全按下才能检测到。

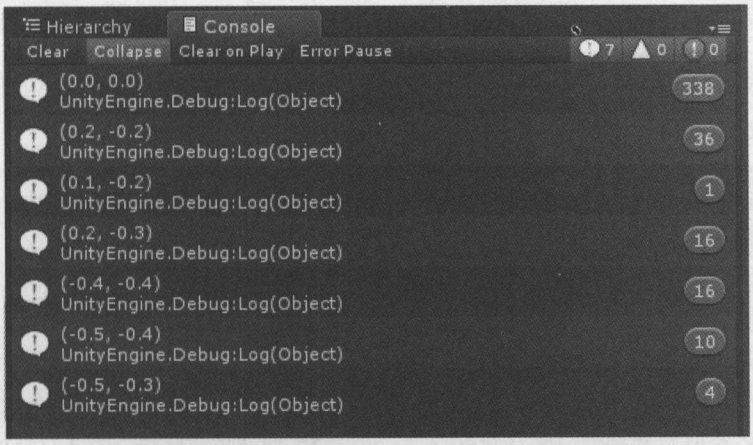
2. 滑动功能

Vive 手柄触摸板可以在特定环境下进行左右或者上下滑动来与物体进行交互，这里我们讲解如何调用它的触摸板滑动功能。

在 Update() 函数中写入如下代码：

```
Vector2 FingerPosition= device.GetAxis (Valve.VR.EVRButtonId.k_EButton_Axis0);
Debug.Log (FingerPosition);
```

保存之后点击运行，在触摸板上轻轻滑动手指，控制台就会输出以下内容。



上图中（0.0,0.0）表示默认是在触摸板中心点，当我们移动手指时，触摸板就会实时记录下当前手指在触摸板上的位置。

触摸板为平面坐标系，竖直握住手柄后水平最右端是 X 轴最大值 +1，竖直最上端是 Y 轴最大值 +1。

以上就是 Vive 触摸板的滑动功能。我们来总结一下：Vive 触摸板滑动功能调用 k_EButton_Axis0 来实现，触摸板 X 轴和 Y 轴的滑动范围都是（−1，+1）。

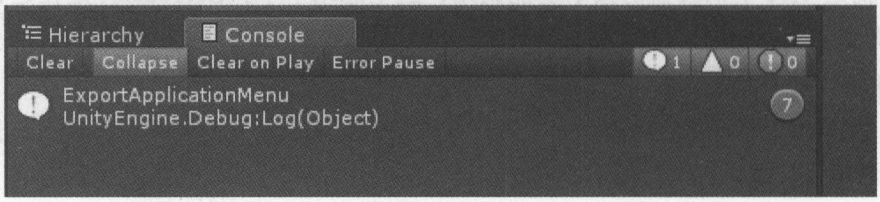
15.7 功能开发之功能键（ApplicationMenu）

Vive 手柄的功能键是指在触摸板上边的一个键，在 Vive 游戏开发中一般用来弹出游戏菜单等。

我们继续在 Update 函数中进行功能调用，在 Update 中写入以下代码并点击运行。

```
if (device.GetPressDown (Valve.VR.EVRButtonId.k_EButton_ApplicationMenu))
{
    Debug.Log ("ExportApplicationMenu");
}
```

当我们按下功能键的时候，控制台输出了如下内容。



类似于功能键这种只需要按一次就能响应事件的，我们都用 Down 函数来执行。

15.8 功能开发之手柄振动（TriggerHapticPulse）

在使用 Vive 手柄进行一些场景交互的时候往往会伴随着手柄的振动，用来提示使用者当前事件已经处于可触发状态。

在 Vive 基本设备调用配置正常之后，在脚本中写入以下代码并保存：

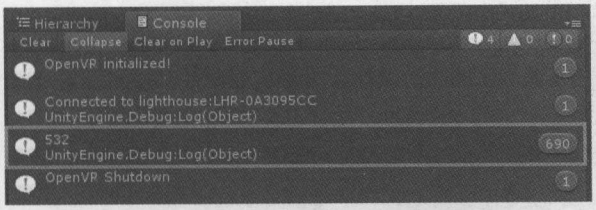
```
void Update ()
{
    device.TriggerHapticPulse (500,Valve.VR.EVRButtonId.k_EButton_Axis0);
        Debug.Log (valve+Valve.VR.EVRButtonId.k_EButton_Axis0);
}
```

其中，device.TriggerHapticPulse() 就是控制手柄振动的函数，该函数有两个参数，第一个是振动的幅度大小，数值为 1 ～ 3999，第二个是相关的振动区域。手柄目前的可振动区域分别是手柄整体和触摸板（Touchpad）处，此处我们选用手柄触摸板为振动区域（如果忽略第二个参数，则默认为手柄整体振动），触摸板在 Vive 底层的 ID 为 32，如下图所示。

所以该代码在运行之后控制台会输出 532。接下来点击运行进行测试。

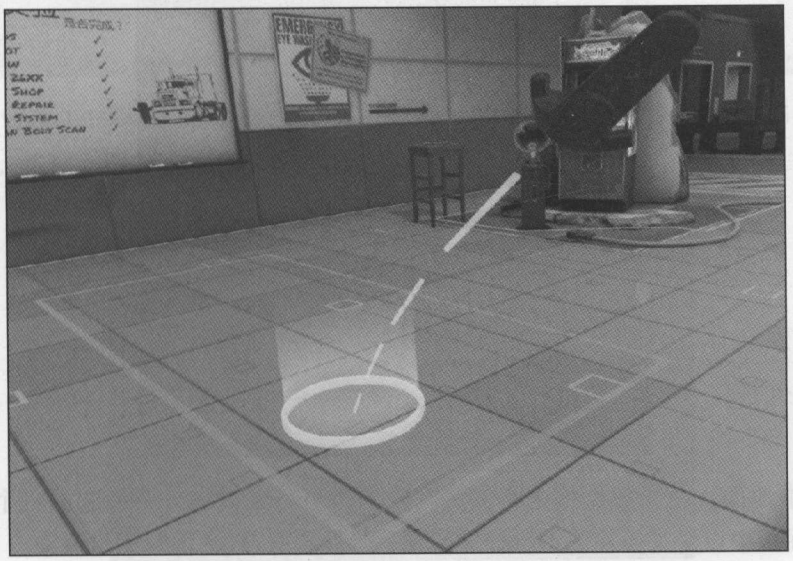
此时，手柄便处于振动状态。

```
public enum EVRButtonId
{
    k_EButton_System = 0,
    k_EButton_ApplicationMenu = 1,
    k_EButton_Grip = 2,
    k_EButton_DPad_Left = 3,
    k_EButton_DPad_Up = 4,
    k_EButton_DPad_Right = 5,
    k_EButton_DPad_Down = 6,
    k_EButton_A = 7,
    k_EButton_Axis0 = 32,
    k_EButton_Axis1 = 33,
    k_EButton_Axis2 = 34,
    k_EButton_Axis3 = 35,
    k_EButton_Axis4 = 36,
    k_EButton_SteamVR_Touchpad = 32,
    k_EButton_SteamVR_Trigger = 33,
    k_EButton_Dashboard_Back = 2,
    k_EButton_Max = 64,
}
```



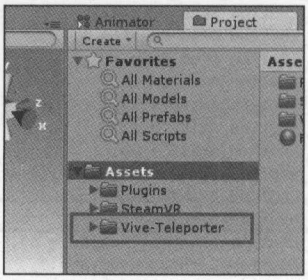
15.9　功能开发之瞬移（Teleporter）

Vive 瞬移就是在场景中使用 Vive 手柄进行短距离快速移动，在官方游戏案例 TheLab 中可以看到，当按下触摸板之后会出现下图所示效果。

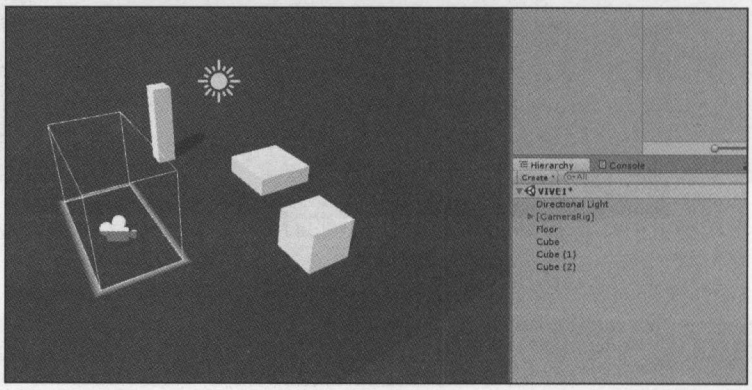


（1）需要下载制作 Vive 瞬移功能时用到的插件，下载地址为 http://pan.baidu.com/s/1qYRDP16，密码为 79g1。

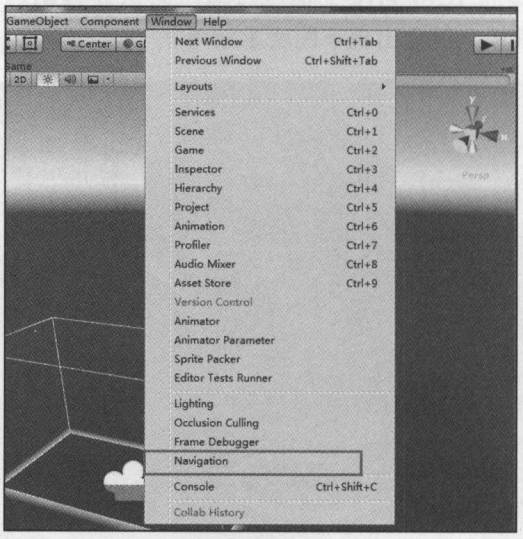
创建一个新的 Unity 工程并导入 SteamVR，将下载好的瞬移插件添加到 Unity 场景中。



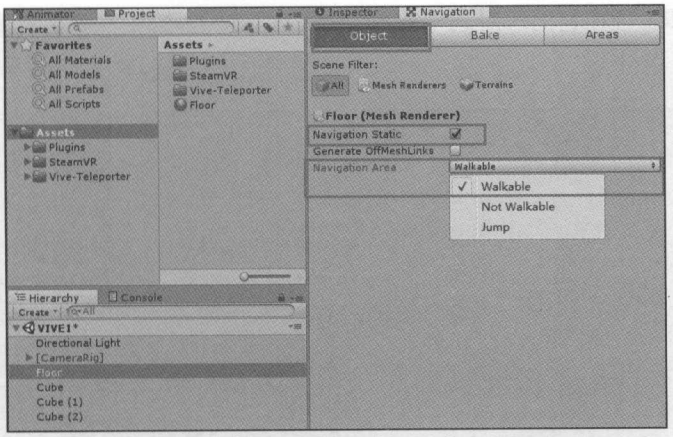
（2）在 Unity 中搭建一个简单的场景。

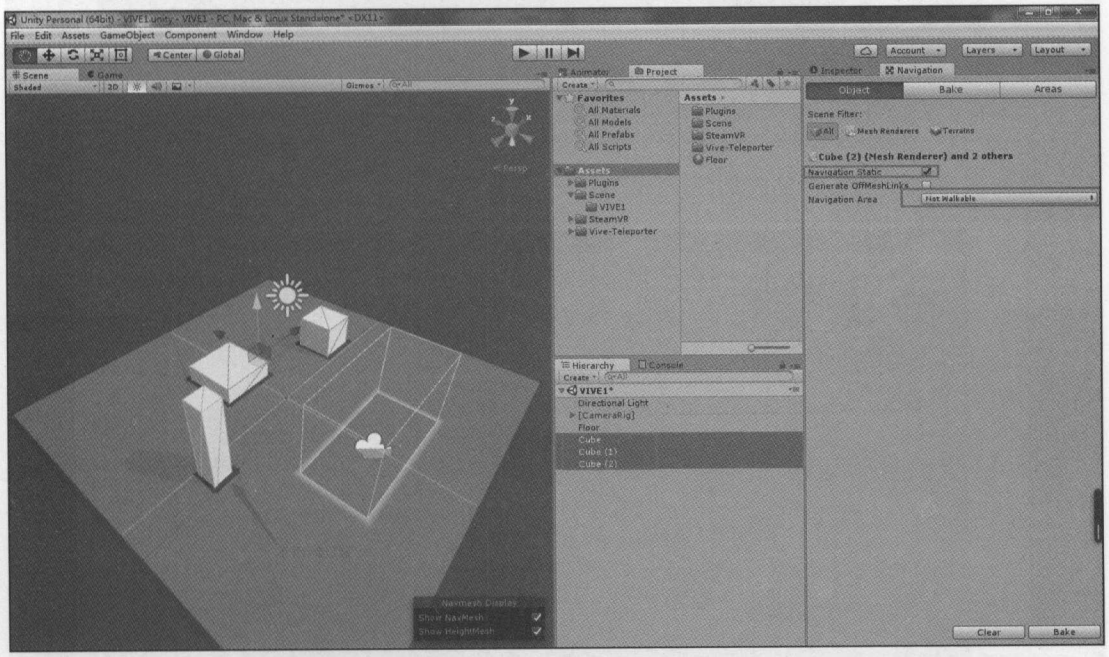


然后点击菜单栏中的 Windows 窗口，选择 Navigation 功能，调出 Unity 导航面板。

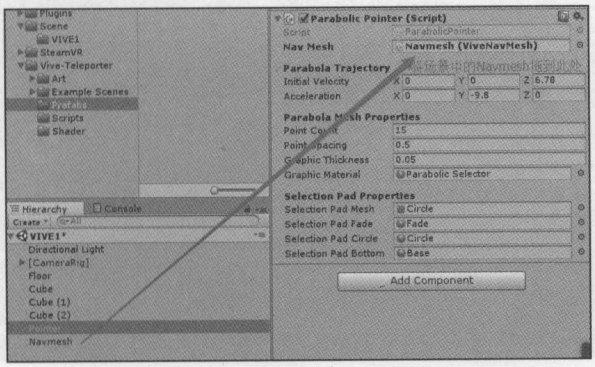


（3）选中地面，在导航面板中将其属性设置为可行走区域，将凸起的三个立方体都选中，在导航面板中设置其属性为不可行走区域，然后点击 Bake 进行烘焙渲染。

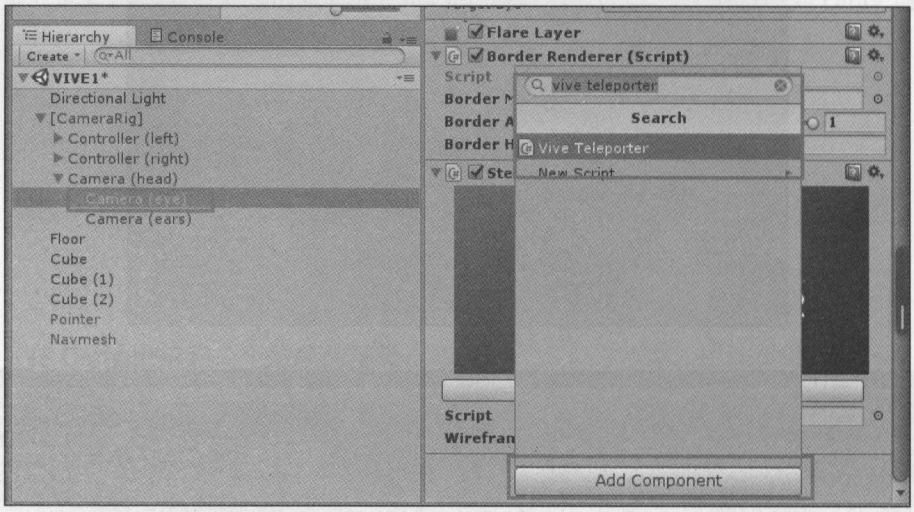




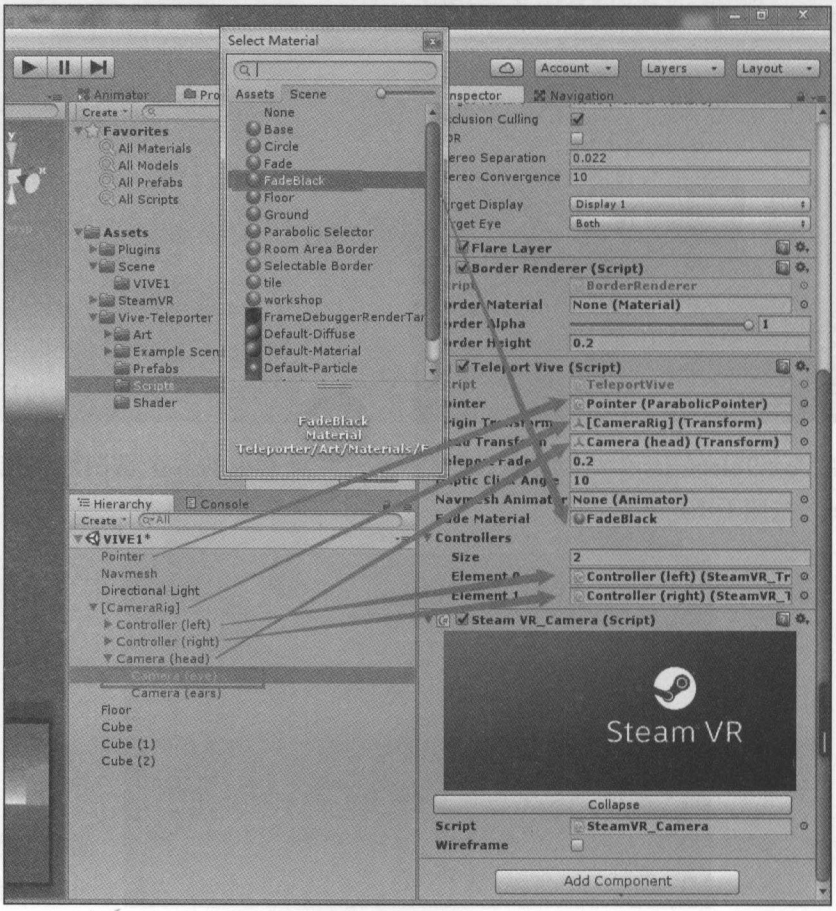
（4）在 Vive-Teleporter 中找到 Prefabs 文件夹，并将其中的两个预制件都拖到场景中。选中 Pointer，按照下图进行操作。

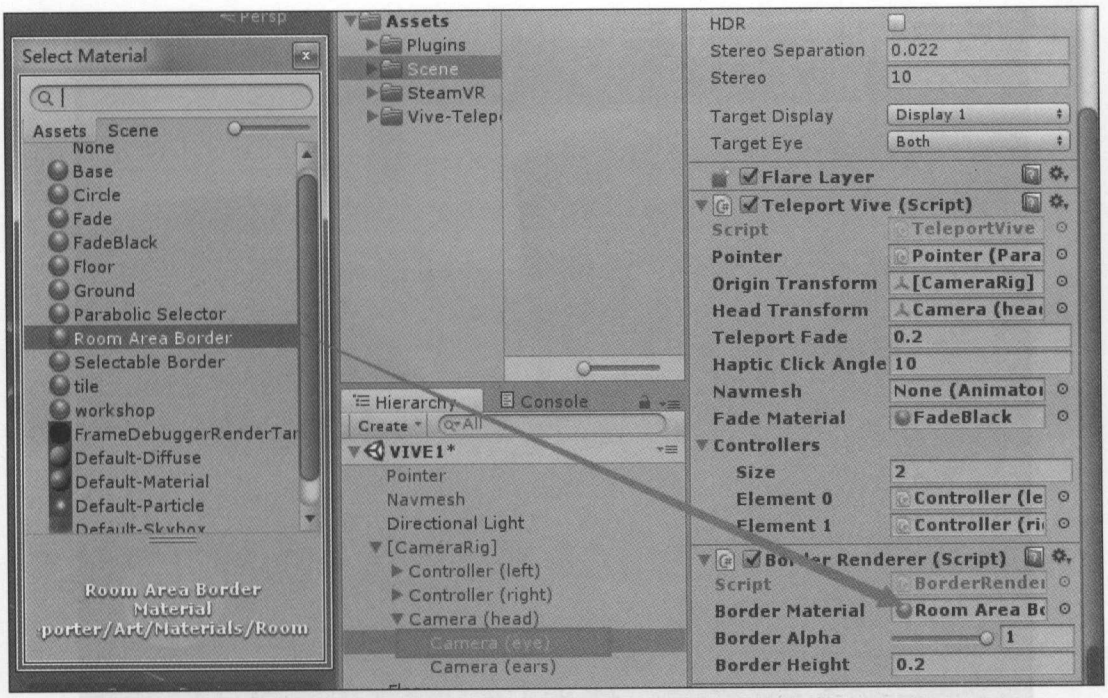


选择 CameraRig，在它的子对象中找到 Camera（eye），选中并在 Inspector 面板中给它添加 Vive Teleporter 组件。

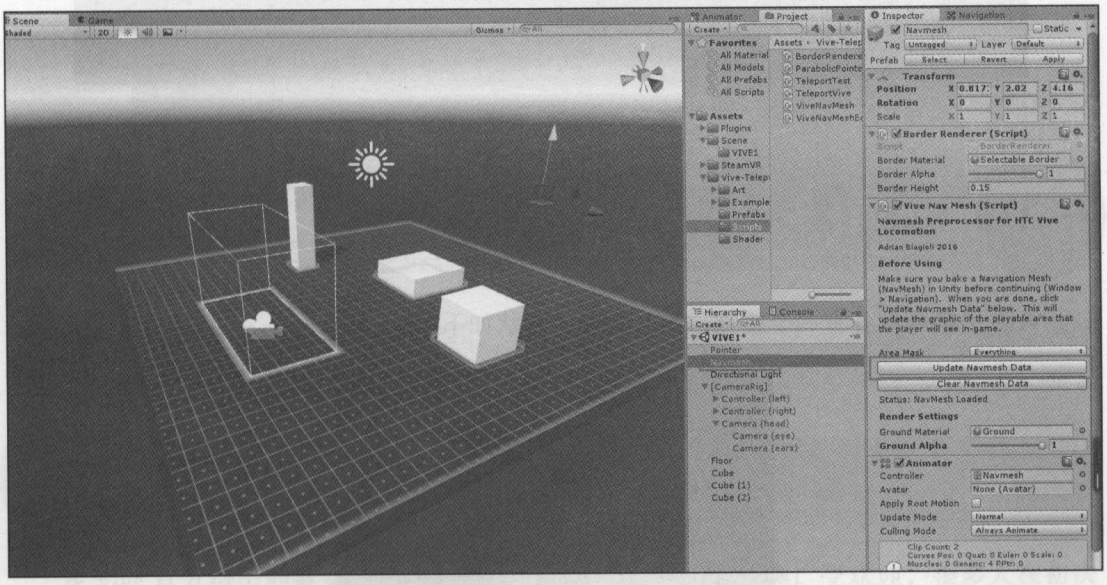


然后将所需要的各组件添加到 Vive Teleporter 上。

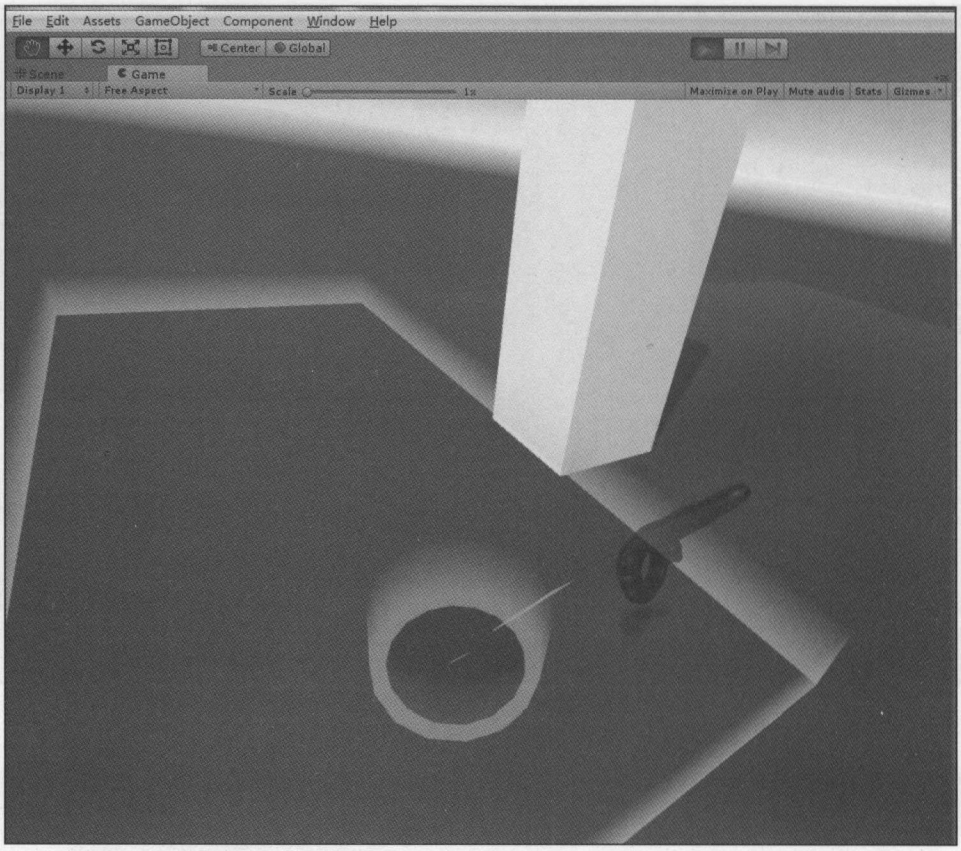




（5）选择 Navmesh，点击 Update Navmesh Data 即可看到下图效果。

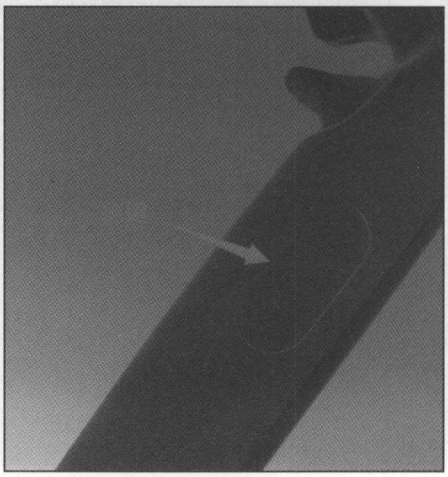


最后点击运行进行测试，效果如下图。



15.10 功能开发之左右侧键（Grip）

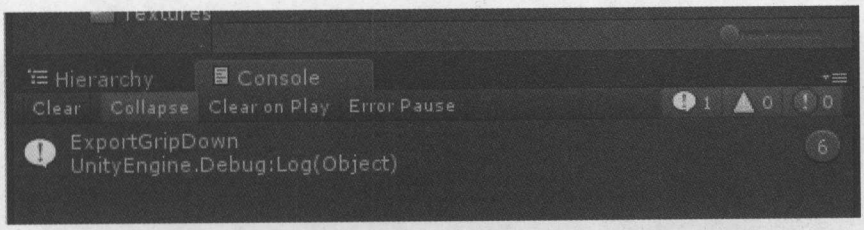
Vive 手柄两侧各有一个按键，称为侧键，形状如下图。



本节继续在 Update 函数中进行功能调用。在 Update 中写入如下代码：

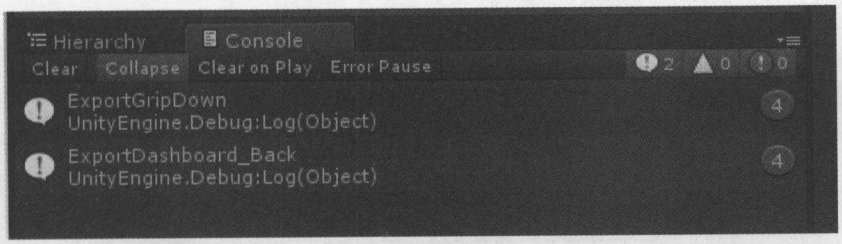
```
if (device.GetPress (Valve.VR.EVRButtonId.k_EButton_Grip))
{
    Debug.Log ("ExportGrip");
}
```

相应的控制台输出内容如下。



我们再调用另外一个函数，让它们同时输出，继续在 Update 中写入代码：

```
if (device.GetPressDown (Valve.VR.EVRButtonId.k_EButton_Dashboard_Back))
{
    Debug.Log ("ExportDashboard_Back");
}
```



可以看到，当我们按下手柄侧键的时候两个函数都有输出，所以在 Vive 手柄事件中，这两个函数触发的是同一个功能。

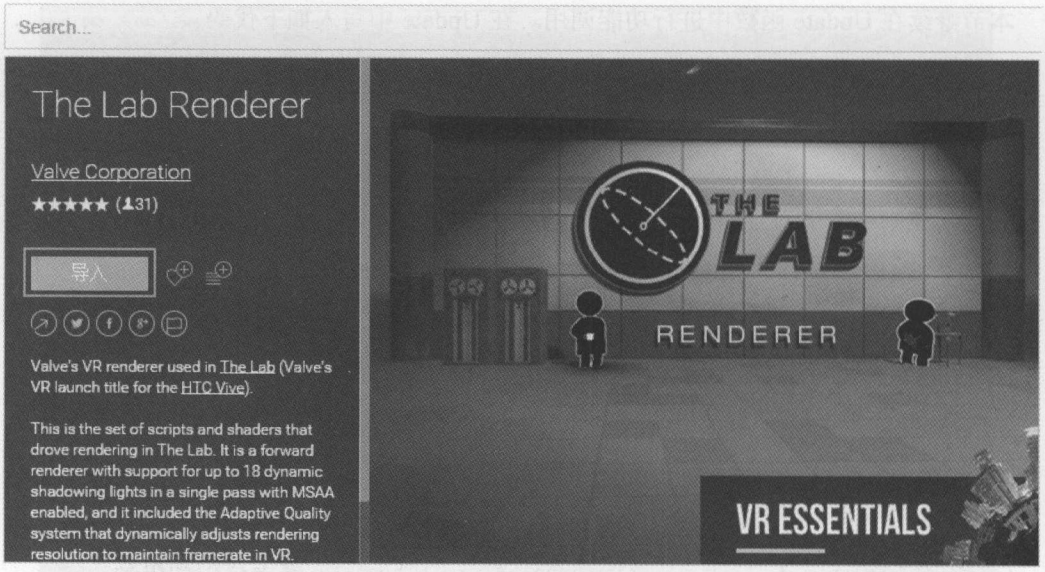
以上就是 Vive 手柄侧键的功能调用。

15.11　功能开发之 The Lab Renderer

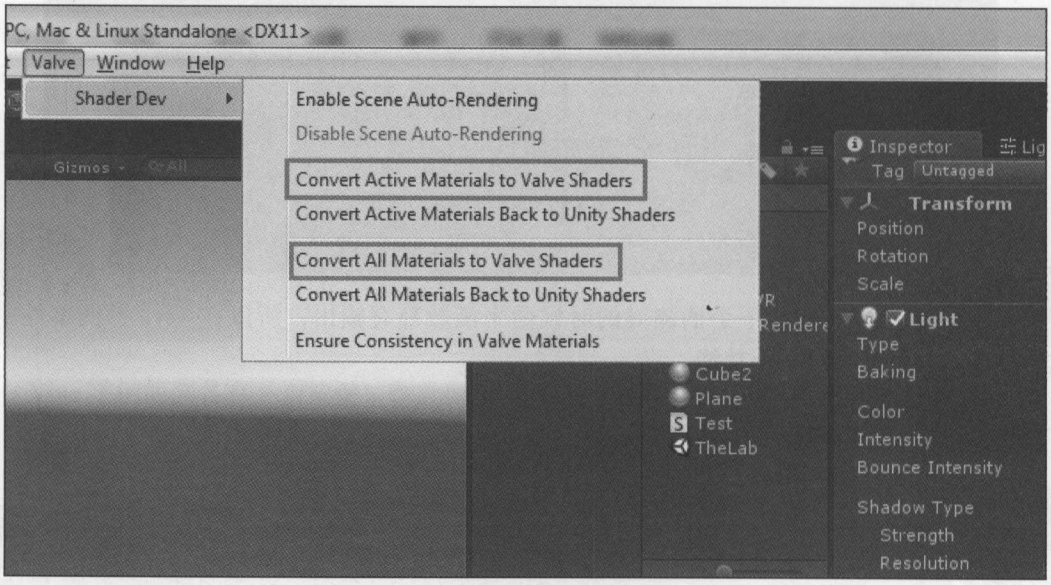
在 Vive 官方给出的案例游戏中有一款叫作 The Lab 的游戏集广受欢迎，很多人在初次接触到 Vive 时就被 The Lab 吸引，而 The Lab 的独特之处除了游戏方式之外还有渲染器。Vive 官方给出了 The Lab 中所有基于 Unity 体验的技术，并且免费供开发者使用。

Unity 目前使用的着色器是多重的，每次光点点亮物体时，几何体又会被渲染一次。The Lab 渲染器在一次前置渲染中最多支持 18 个动态阴影光线。

在 Unity 资源商店中搜索 The Lab Renderer，下载并导入。



导入完成之后选中场景中的主摄像机，并添加脚本 Valve Camera，在当前场景的烘焙光源上添加脚本 Valve Realtime Light，完成之后在菜单栏中会出现 Valve 菜单选项。



点击即可将场景中的材质转换为 Valve 提供的 Shader（代替 Unity5 Standard Shard）。相较于 Unity 提供的渲染设施，The Lab 渲染器具有以下一系列优点：

- 单向前置渲染和多重采样抗锯齿。
- 自适应质量。
- 自定义着色器。
- GPU 刷新。

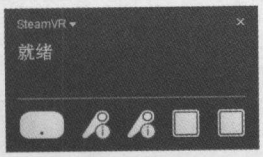
第16章 基于 HTC Vive 实现房间漫游

在 VR 技术普遍成熟之后，随之兴起的一项重要内容就是 VR 虚拟房间漫游，目前国内在制作虚拟房间漫游时主要使用虚幻游戏引擎（UE4）和 Unity 3D 游戏引擎，本节主要使用 Unity 3D 来实现房间漫游。

16.1 准备工作

在开始制作房间漫游之前，需要准备以下内容：

- Unity 3D 游戏引擎，本节使用的版本为 5.3.4f1，建议保持一致。官网下载地址为 https://unity3d.com/cn/get-unity/download/archive?_ga=1.67797329.116705845.1462463231
- 基于 Unity 3D 游戏引擎的 SteamVR Plugs 组件，在 Unity3D 自带的资源商店中即可下载。
- 电脑显卡不低于 GTX970，并且安装了 Steam 游戏平台以及平台内的 SteamVR 组件。
- 确保 SteamVR 正常工作，并设定了房间范围（如有疑问，可参照前面的章节）。



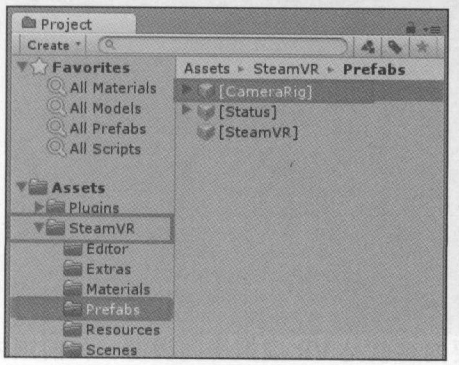
- 一套房间模型。

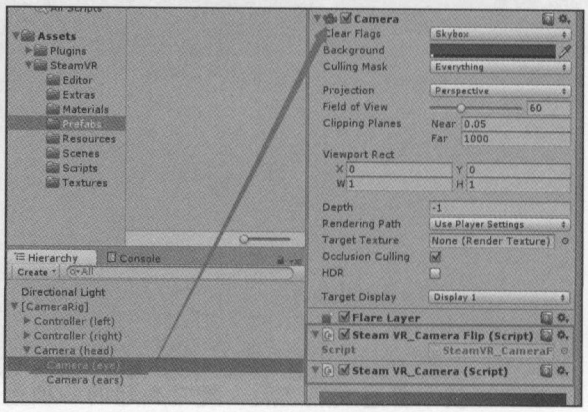
16.2 基本平台搭建

新建一个 Unity 3D 工程，命名为 VIVEHouse，打开资源商店搜索下载 SteamVR Plugs

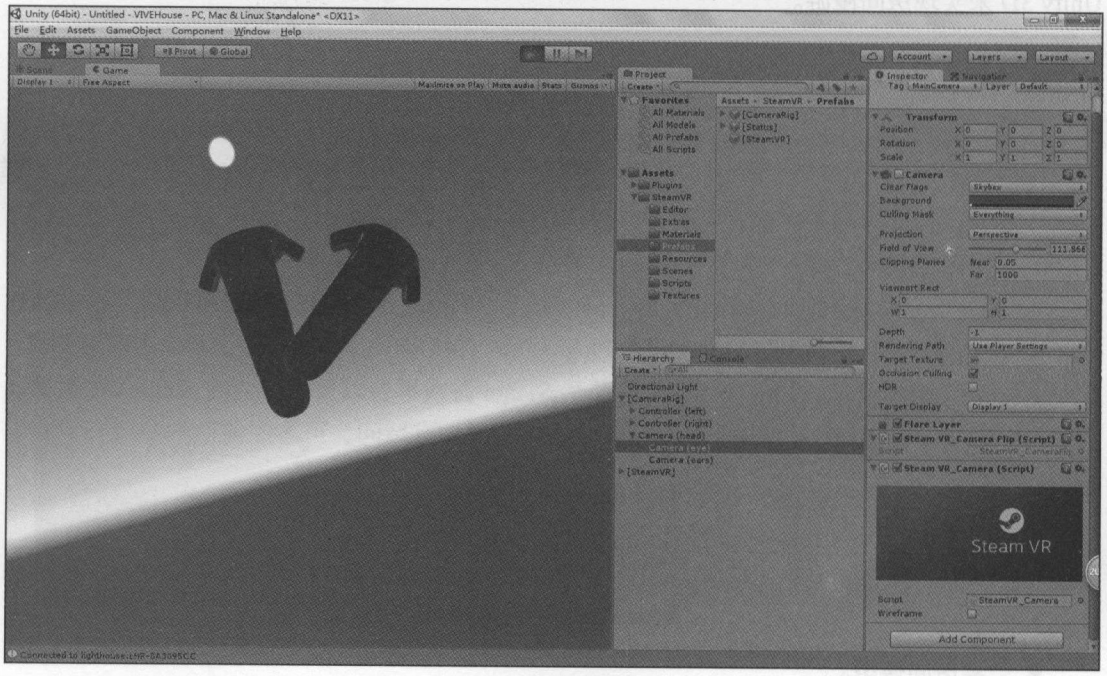
并导入。SteamVR 组件是 Valve 公司专为 Unity 3D 开发 Vive 系列 VR 游戏而推出的，通过该组件我们可以制作出各种适用于 HTV Vive 的游戏应用。

导入 SteamVR Plugs 完成之后，将场景中的主摄像机删除，同时把如下图所示的预制件拖到场景中，该预制件包含 Vive 头盔显示器（下边简称头显）所适用的专用摄像机。





点击 Unity 3D 运行按钮，即可通过头显摄像机看到 Unity 3D 的虚拟世界。

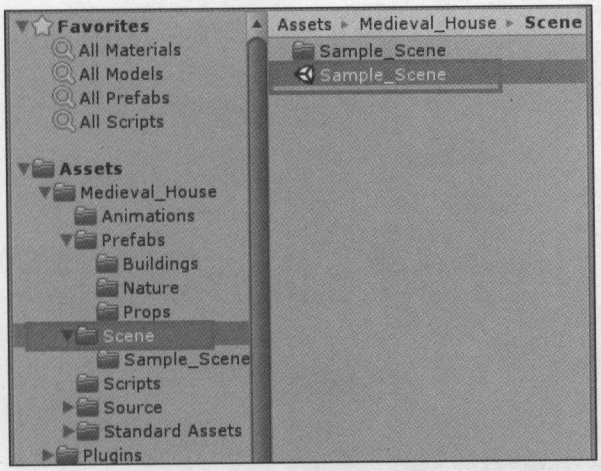


16.3 导入房间模型

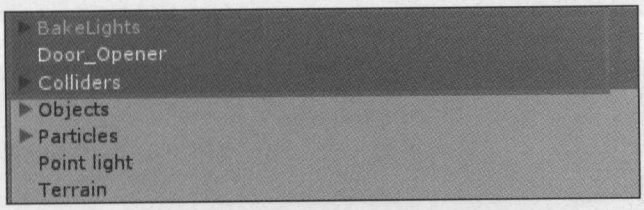
将准备好的房间模型导入 Unity 3D 中，本节导入的模型格式为 .unitypackage，除此之

外，Unity 3D 还支持“.fbx”和“.obj”格式的模型。

打开模型自带的场景并另存一份，更名为 VIVEHouse。



将场景中的如下游戏对象删除。



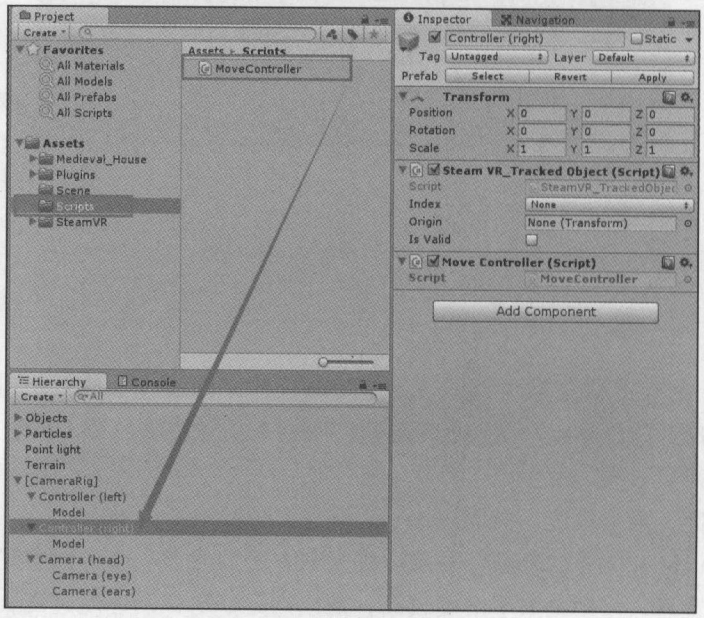
16.4 基于 Vive 手柄的场景移动

一般的 Vive 游戏位移分为两种。一种是跳跃性的，即通过手柄扳机键（Trigger）将摄像头移动到当前选择的位移点，从而实现移动。该位移方法主要依赖 Unity 3D 的 Navigation 渲染地形从而获得可移动的位置点，其优点在于画面稳定、眩晕感小，缺点是对于当前地形的平整度要求较高，地形坡度不能超过 5°，否则无法识别到位置点。另一种就是通过 Vive 手柄触摸板（TouchPad）滑动来进行平滑位移，该方法的优点在于可以实现复杂场景地形漫游，不用规定位置点，移动画面平滑度较高，缺点则是如果移动速度过高，则会出现眩晕等不适应症状。本节使用第二种方式来实现场景位移。

首先将 SteamVR 中的头显摄像机拖到场景中，位置如下，并进行调整。

新建文件夹并命名为 Scripts，在其中新建一个 C# 脚本，命名为 MoveController，把该脚本拖到场景对象 [CameraRig] 下的 Controller(right) 上，然后双击打开并进行编辑。





脚本编辑的第一步是获取当前发送位移请求的设备（即获取手柄）。

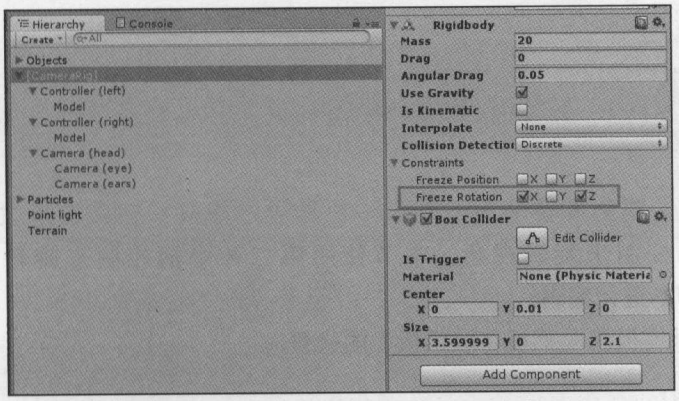
```
public class MoveController : MonoBehaviour
{
  SteamVR_TrackedObject trackedObject;
  void Start ()
    {
    trackedObject = this.GetComponent<SteamVR_TrackedObject> ();
    }
```

```
void Update ()
    {
        var device = SteamVR_Controller.Input ((int)trackedObject.index);
}
... ...
}
```

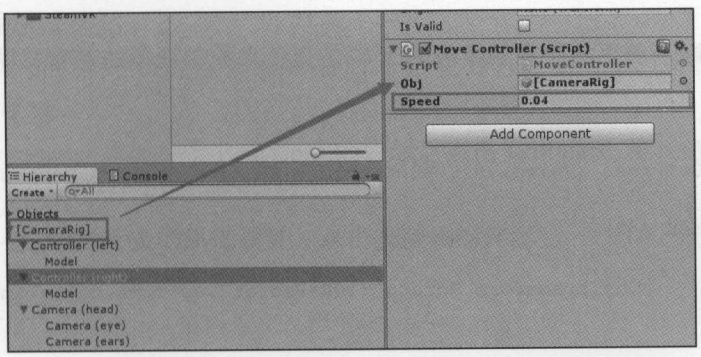
接着获取手柄触摸板按下事件，同时给当前物体一个向前的位移以及速度。

```
public class MoveController : MonoBehaviour
{
public GameObject Obj;
    public float Speed=0.4f;
void Update ()
    {
 ... ...
if (device.GetPress (Valve.VR.EVRButtonId.k_EButton_SteamVR_Trigger))
      {
            Obj.transform.position += transform.forward * Speed;
      }
}
```

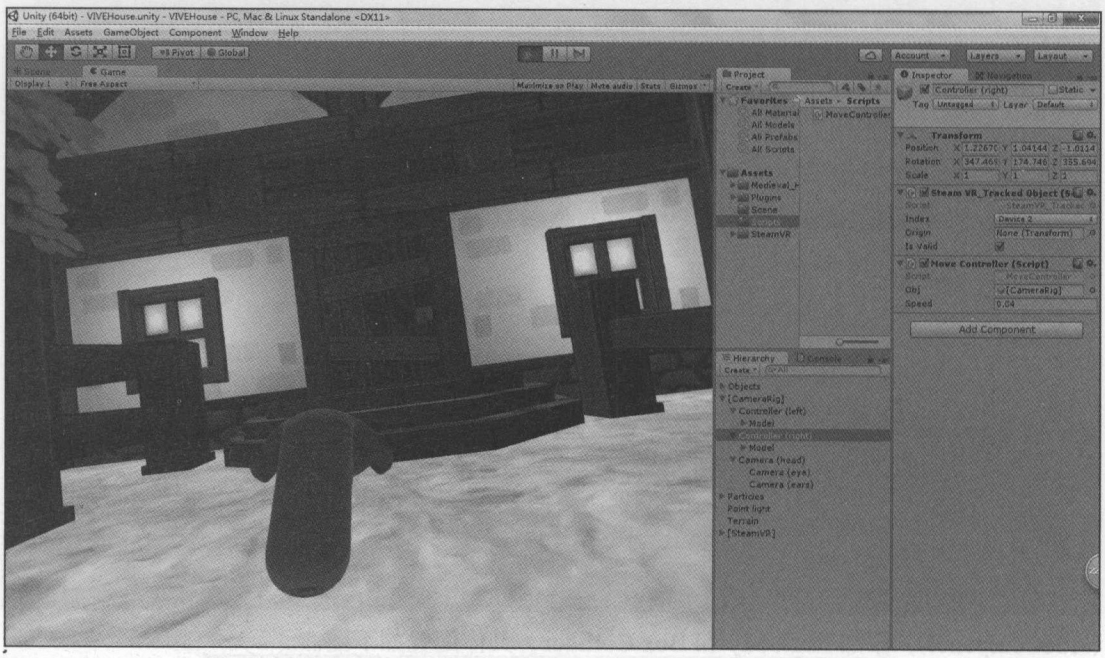
保存代码并退出，选中场景中的 [CameraRig]，并且在 Inspector 面板中点击添加组件（AddComponet），添加 Rigibody 和 Box Collider，具体配置如下图。



选择 Controlle（right）将当前速度更改为自身所适应的合适速度，在空出的 Obj 栏中拖入 [CameraRig] 对象。



接下来点击运行进行测试，扣动扳机键时，角色就可以在场景中移动了。



16.5 基于 Vive 手柄的场景旋转

在 Vive 场景中视角旋转分为两种，一种是通过获取头盔旋转来改变当前视角，另一种是通过手柄控制来旋转视角，本节使用第二种手柄控制来旋转视角。双击打开 MoveController 脚本进行编辑。

首先使手柄触摸板获取手指在触摸板上的位置。

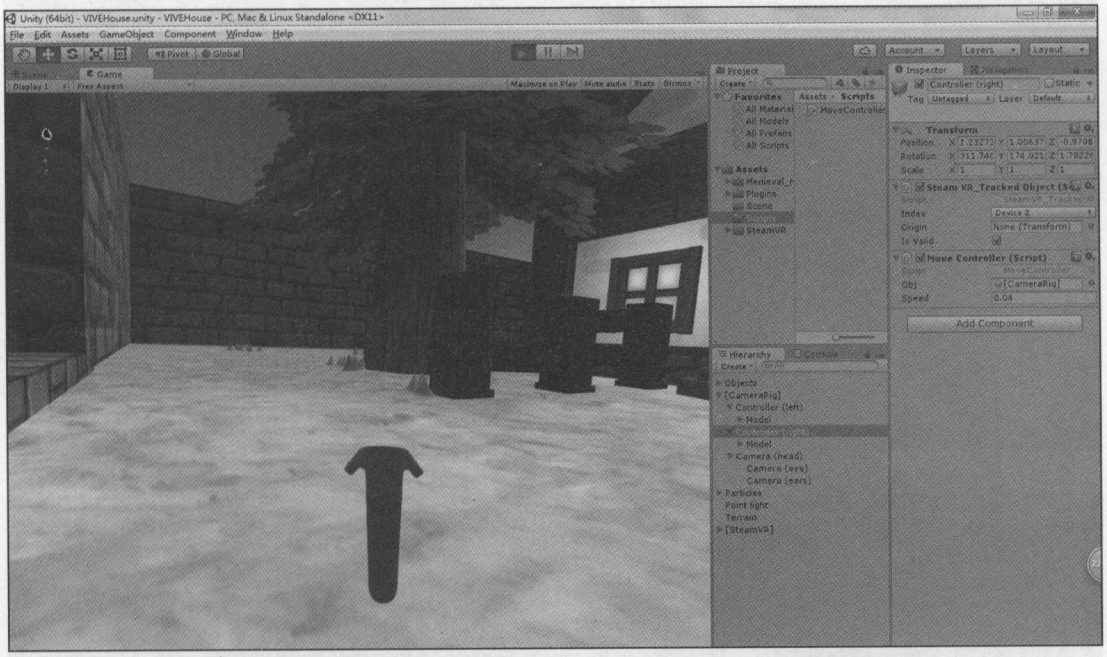
```
void Update ()
{
  float PosX =- device.GetAxis (Valve.VR.EVRButtonId.k_EButton_Axis0).x;
… …
}
```

然后通过判断当前手指所在触摸板的 X 轴的正负来控制视角的左右移动。

```
void Update ()
{
if (device.GetPress (Valve.VR.EVRButtonId.k_EButton_SteamVR_Touchpad))
{
      if (PosX < 0)
      {
            Obj.transform.Rotate (100*Vector3.up*Time.deltaTime);
      }
```

```
        else if (PosX > 0)
        {
                Obj.transform.Rotate (100*Vector3.down*Time.deltaTime);
        }
    }
... ...
}
```

保存代码并退出，然后点击运行测试效果，分别按下触摸板的左侧和右侧以观察场景中视角的变化。



16.6 手柄与场景物体的交互

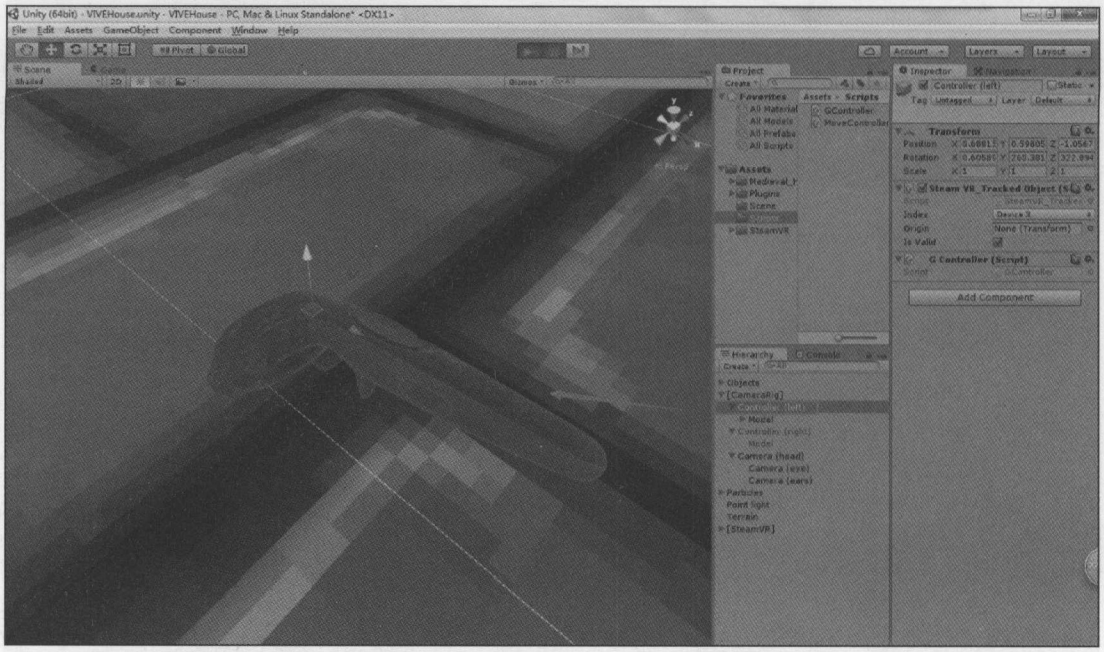
在 Scripts 文件中新建一个 C# 脚本并命名为 GController，拖到 Controller（left）对象上。

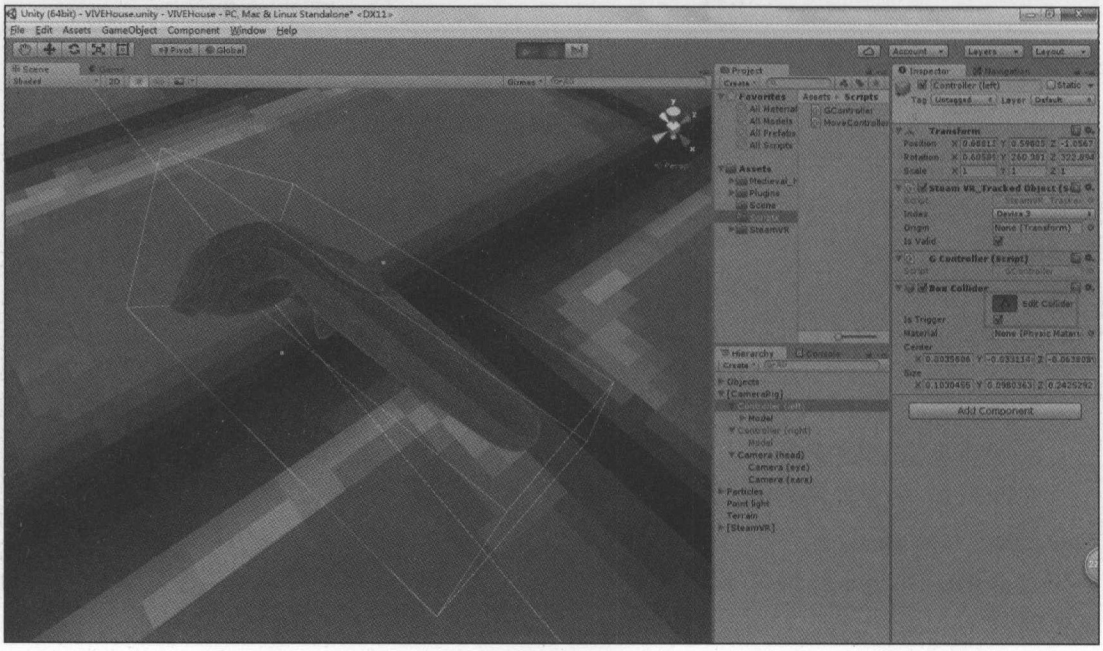
1. 给手柄添加碰撞器

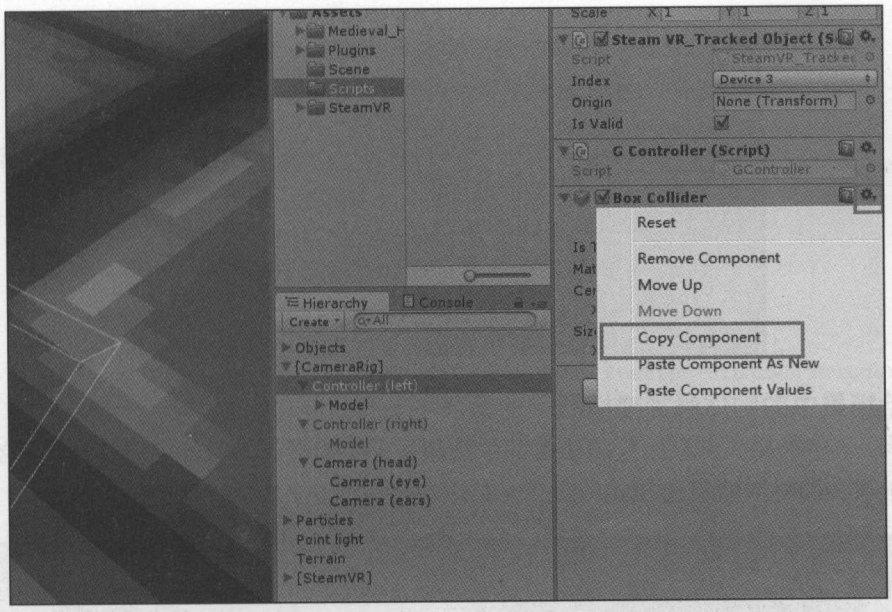
在编辑脚本之前首先点击运行场景然后点击暂停，并且在 Scene 场景中居中显示 Controller（left）手柄。

点击 Add Componet 添加一个 Box Collider，然后点击 Collider，将其碰撞范围缩小到合适范围。

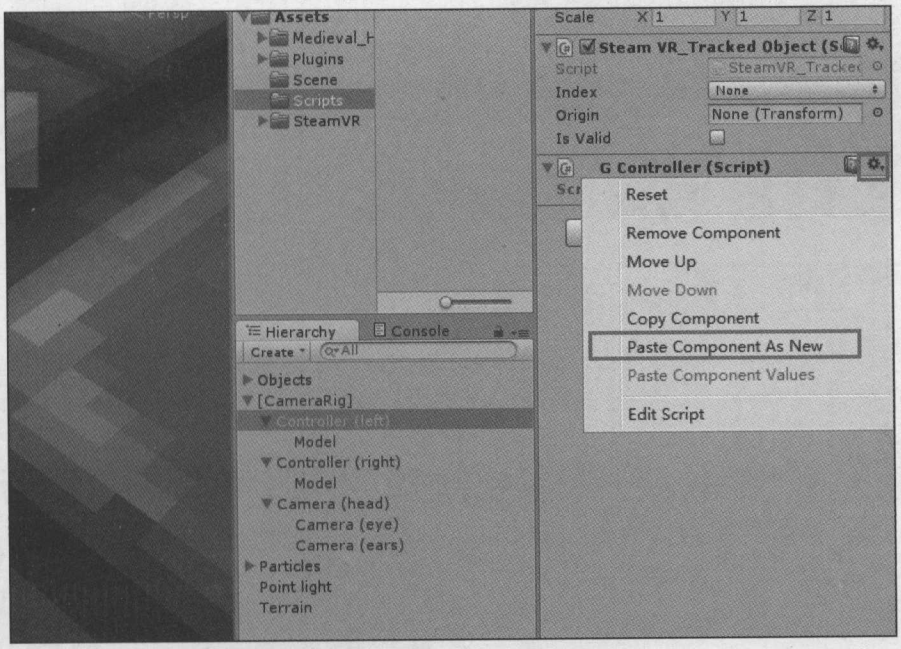
接下来点击 Collider 上边的属性设置，点击复制该组件。


Unity (64bit) - VIVEHouse.unity - VIVEHouse - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Pivot Global
Account Layers Layout
Scene
Shaded 2D
Gizmos
Project
Create
Favorites
All Material
All Models
All Prefabs
All Scripts
Assets
Medieval_P
Plugins
Scene
Scripts
SteamVR
Assets > Scripts
GController
MoveController
Hierarchy
Console
Create
Objects
[CameraRig]
Controller (left)
Model
Controller (right)
Model
Camera (head)
Camera (eye)
Camera (ears)
Particles
Point light
Terrain
[SteamVR]
Inspector
Navigation
Controller (left)
Static
Tag Untagged
Layer Default
Transform
Position X 0.6881 Y 0.5905 Z -1.0547
Rotation X 0.6058 Y 260.381 Z 322.894
Scale X 1 Y 1 Z 1
Steam VR_Tracked Object (S
Index Device 3
Origin None (Transform)
Is Valid
G Controller (Script)
Add Component


Unity (64bit) - VIVEHouse.unity - VIVEHouse - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Pivot Global
Account Layers Layout
Scene
Shaded 2D
Gizmos
Project
Create
Favorites
All Material
All Models
All Prefabs
All Scripts
Assets
Medieval_P
Plugins
Scene
Scripts
SteamVR
Assets > Scripts
GController
MoveController
Hierarchy
Console
Create
Objects
[CameraRig]
Controller (left)
Model
Controller (right)
Model
Camera (head)
Camera (eye)
Camera (ears)
Particles
Point light
Terrain
[SteamVR]
Inspector
Navigation
Controller (left)
Static
Tag Untagged
Layer Default
Transform
Position X 0.6881 Y 0.5905 Z -1.0547
Rotation X 0.6058 Y 260.381 Z 322.894
Scale X 1 Y 1 Z 1
Steam VR_Tracked Object (S
Index Device 3
Origin None (Transform)
Is Valid
G Controller (Script)
Box Collider
Edit Collider
Is Trigger
Material None (Physic Materi
Center
Size
Add Component

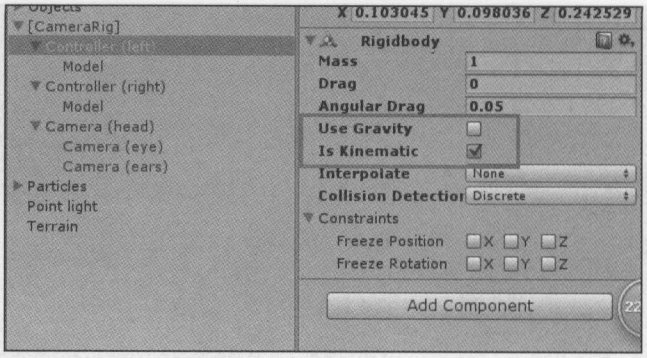


停止场景运行，然后再次选中 Controller（left），将刚才复制的组件重新添加上去。



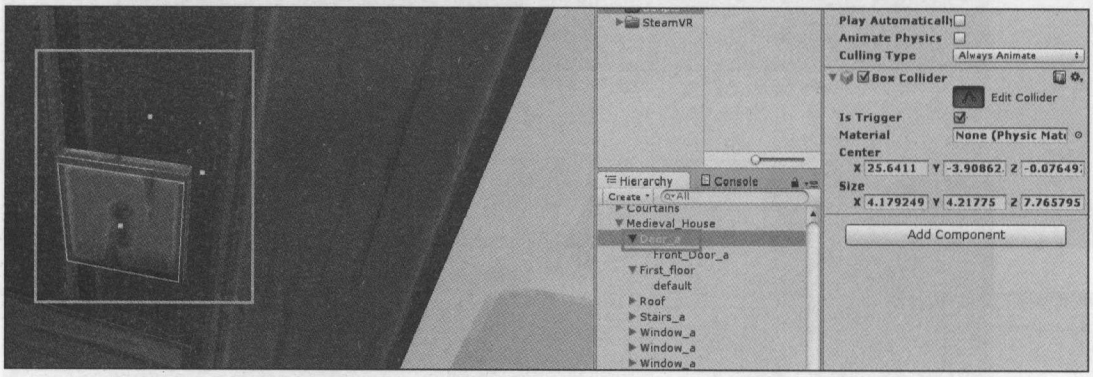
这里需要了解的是，对于 Unity 3D 在运行状态下所编辑的内容，停止运行 Unity 3D 场景之后都会清除掉，所以一般对于场景中游戏对象的编辑都是在非运行状态下进行的。这里需要对 Vive 手柄添加一个 Box 碰撞器，但是手柄只有在运行状态下才会出现，在非运行状态下是不可见状态，所以我们需要在运行时给手柄添加组件，同时将其复制出来，然后待场景停止运行后再将该组件添加上去。

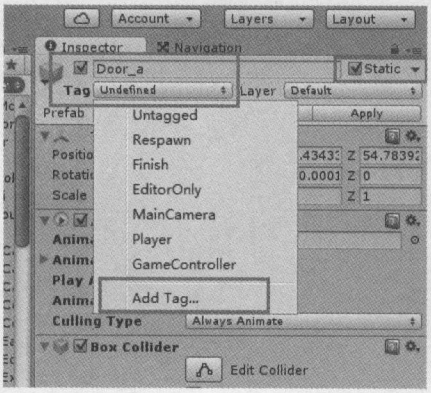
接下来再给手柄添加一个刚体（Rigibody），并按照下图进行配置。

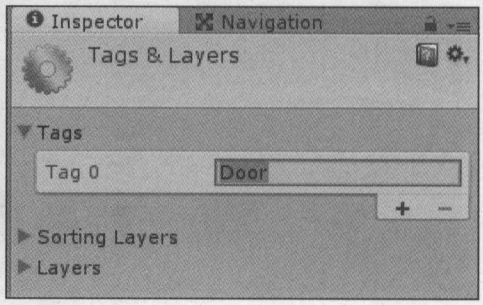


2. 给场景中的物体添加碰撞器

选择场景中的门，并且给门的钥匙部位添加一个碰撞器，同时给门设定一个标签（tag），命名为 Door，同时取消勾选 Static。







双击打开 GController 脚本并进行编辑。

首先获取需要交互的手柄：

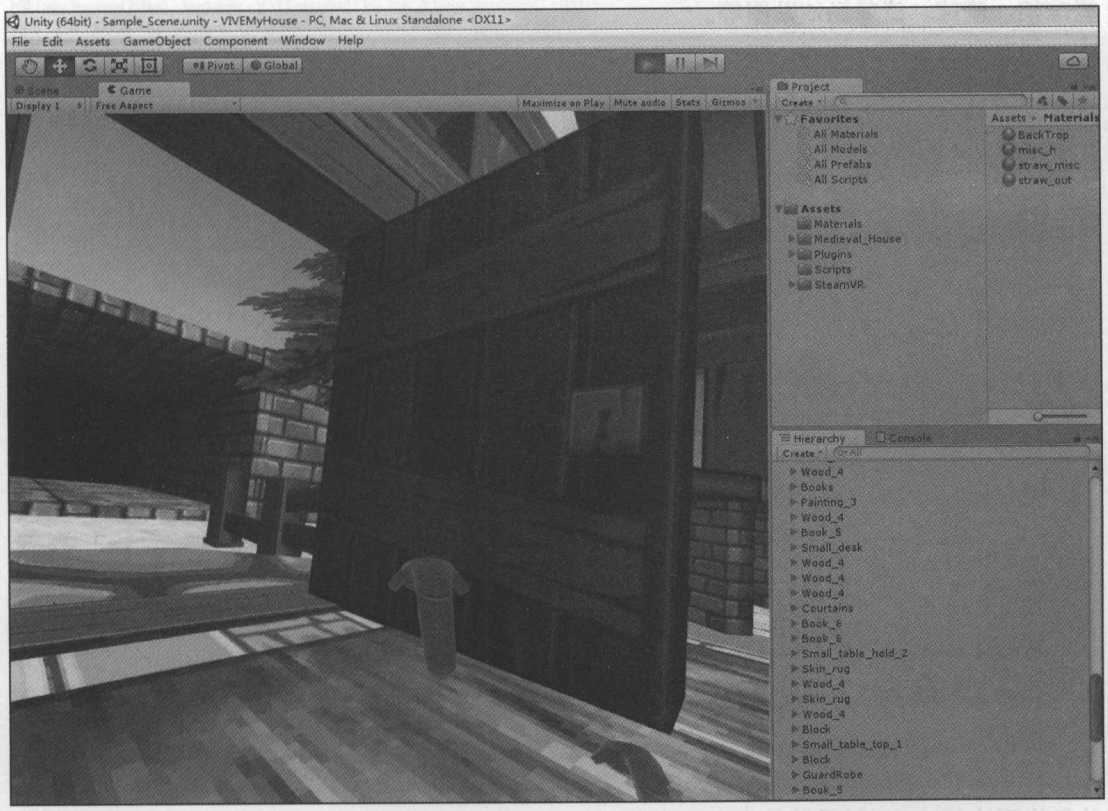
```
public class GController : MonoBehaviour
{
        SteamVR_TrackedObject trackedObject;
void Start ()
        {
```

```
                trackedObject = this.GetComponent<SteamVR_TrackedObject> ();
        }
... ...
}
```

然后使用触发器来判断手柄是否碰到场景中的物体：

```
void OnTriggerStay(Collider other)
{
    var device = SteamVR_Controller.Input ((int)trackedObject.index);
    if (other.gameObject.tag == "Door")
    // 要碰撞的物体的标签为 Door
    {
    if (device.GetPressDown(Valve.VR.EVRButtonId.k_EButton_SteamVR_Trigger))
    // 扣动扳机键来触发事件
        {
            Animation anim=other.GetComponent<Animation>();
            anim.Play ();// 调用标签为 Door 的物体的动画系统并播放动画
        }
    }
... ...
}
```

保存代码并退出，在场景中找到一个叫作 OpenDoor 的脚本并删除，点击运行测试效果，可以看到，当我们将手柄放到门锁的位置并扣动扳机键的时候门会打开。

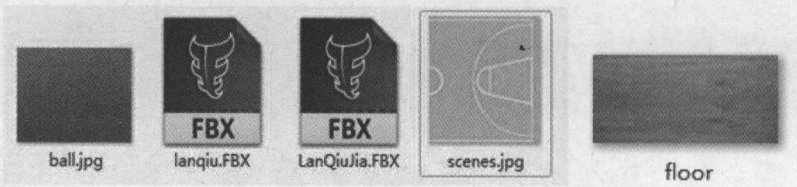


通过以上步骤我们就实现了基于 Unity 3D 的 Vive 场景漫游的制作。

第17章 基于 HTC Vive 开发投篮游戏

17.1 准备素材

本节带领大家制作一个简单的 VR 投篮游戏，在开始之前我们需要准备一些素材，比如篮球架、篮球、篮球场地贴图等。



在 Unity 3D 资源商店下载 HTC Vive 的组件 SteamVR Plugs 并导入。

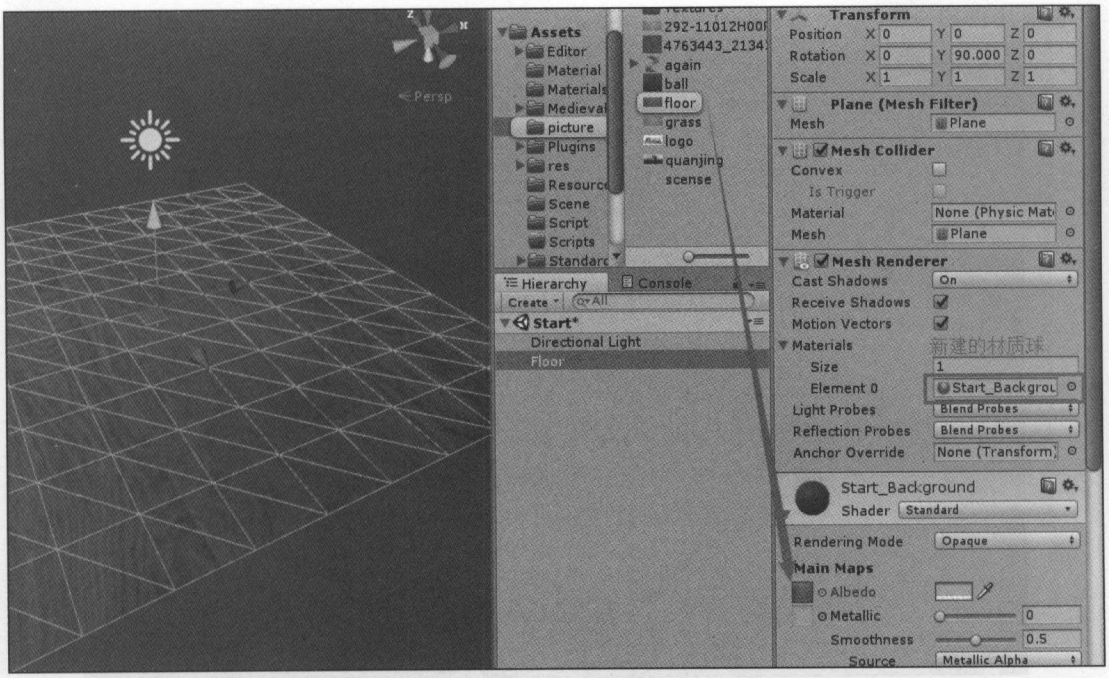
17.2 场景搭建

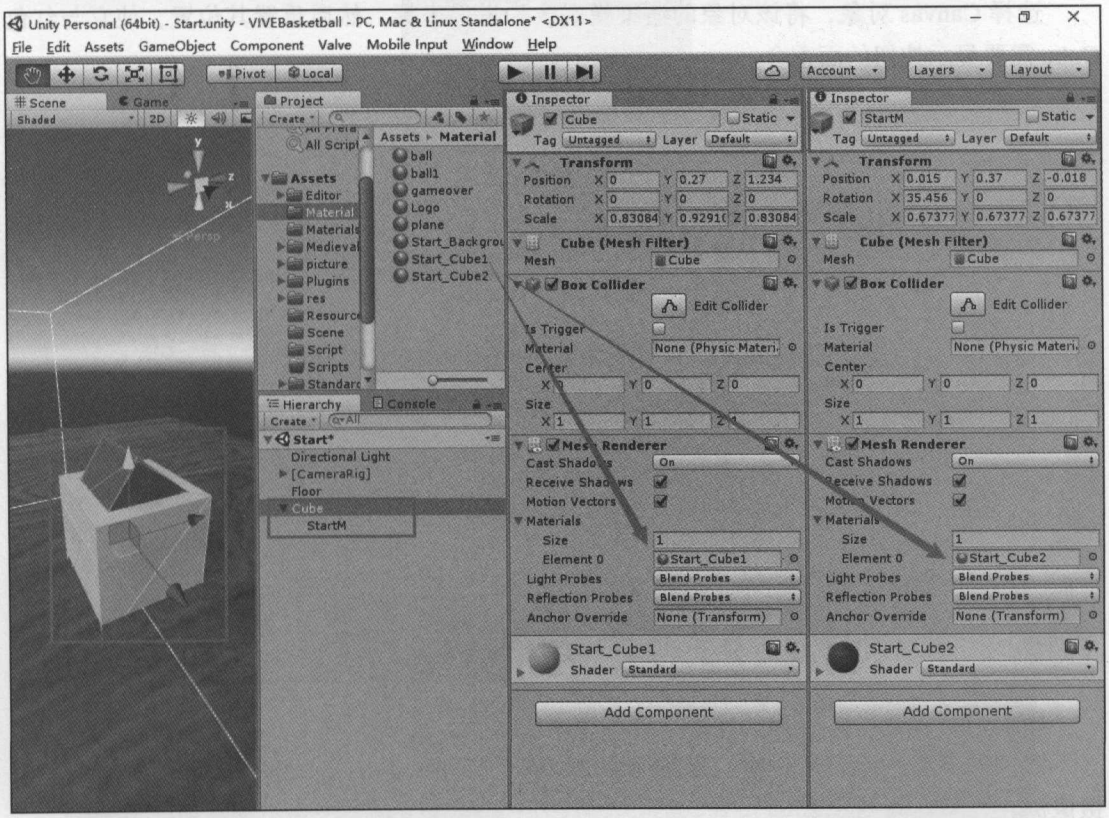
1. 开始界面的制作

在开始界面中，当我们使用手柄触碰到开始按钮之后就会进入投篮的游戏场景中。

在场景中创建一个 Plane，并给 Plane 添加一个材质球，然后将导入的 floor 贴图贴上去。想要在一个物体上添加一张好看的贴图，首先要给该物体添加一个材质球，然后再把贴图附着在材质球上面。

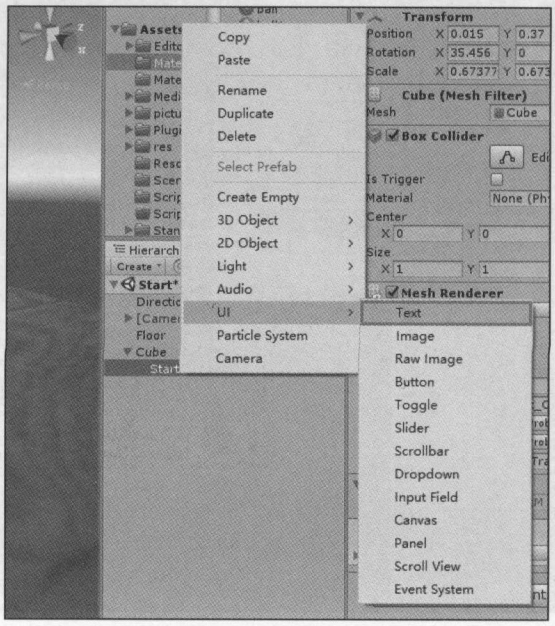
接下来继续创建两个 Cube，作为开始按钮的触发界面。分别给两个 Cube 创建材质球，并且将下边的 Cube 材质球的颜色改为黄色，上边的 Cube 材质球的颜色改为红色，位置关系如图所示。

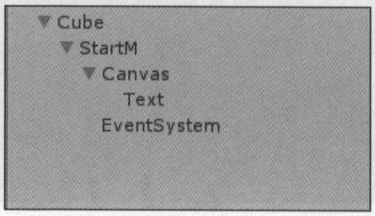




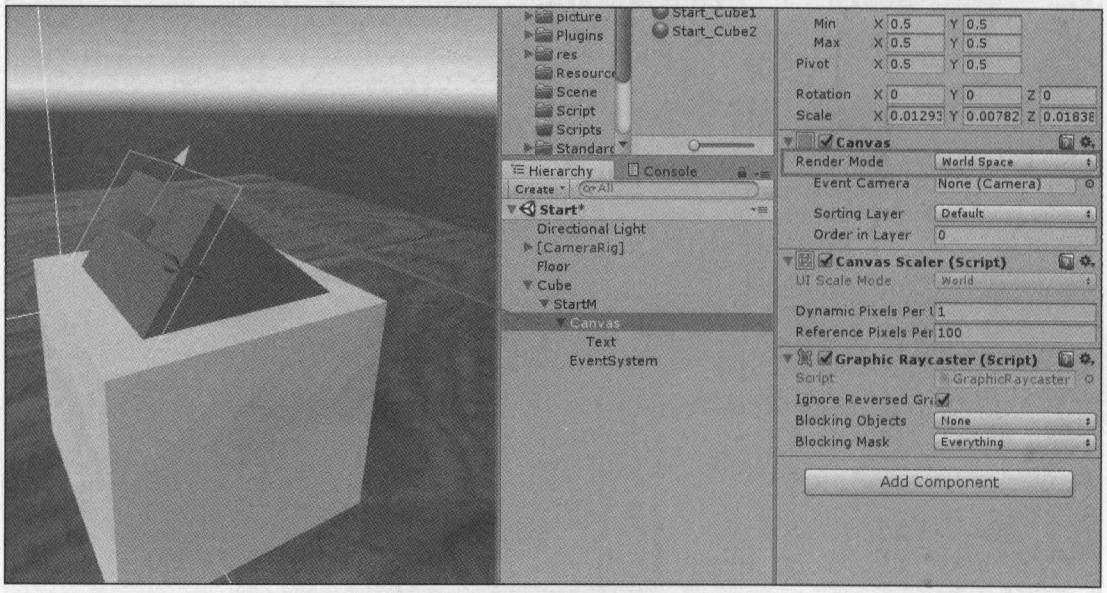
做完这一步后，我们给红色 Cube 添加一个可以用手柄来触发的按钮。选择红色 Cube，

在它下边创建一个 Text，此时在红色 Cube 下边就会多出下图所示的三个对象。



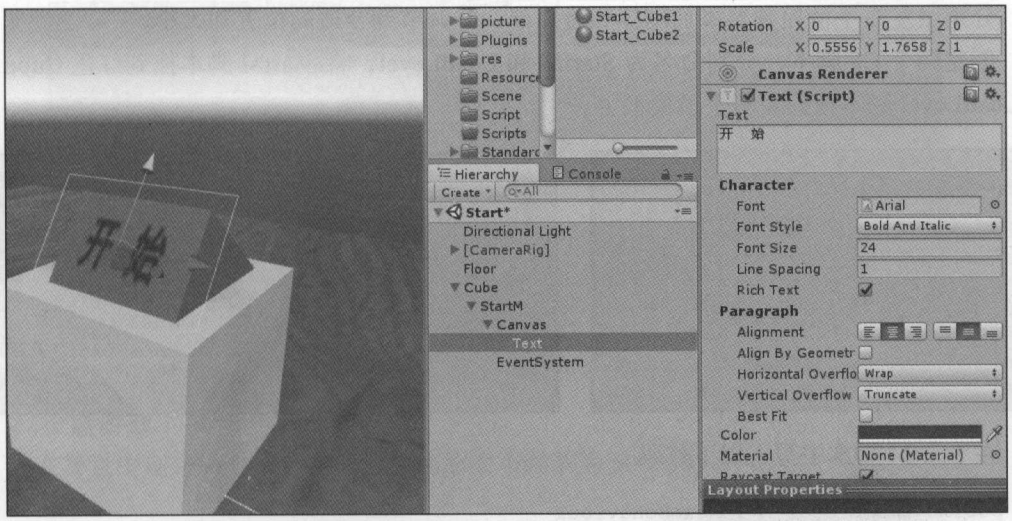


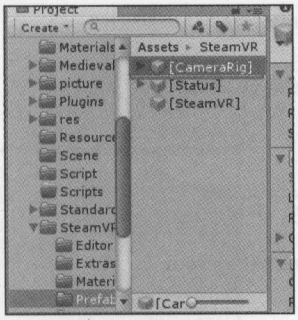
选择 Canvas 对象，将该对象的渲染模式改为世界坐标，然后重置其位置，使它与红色 Cube 需要显示按钮的面重合。

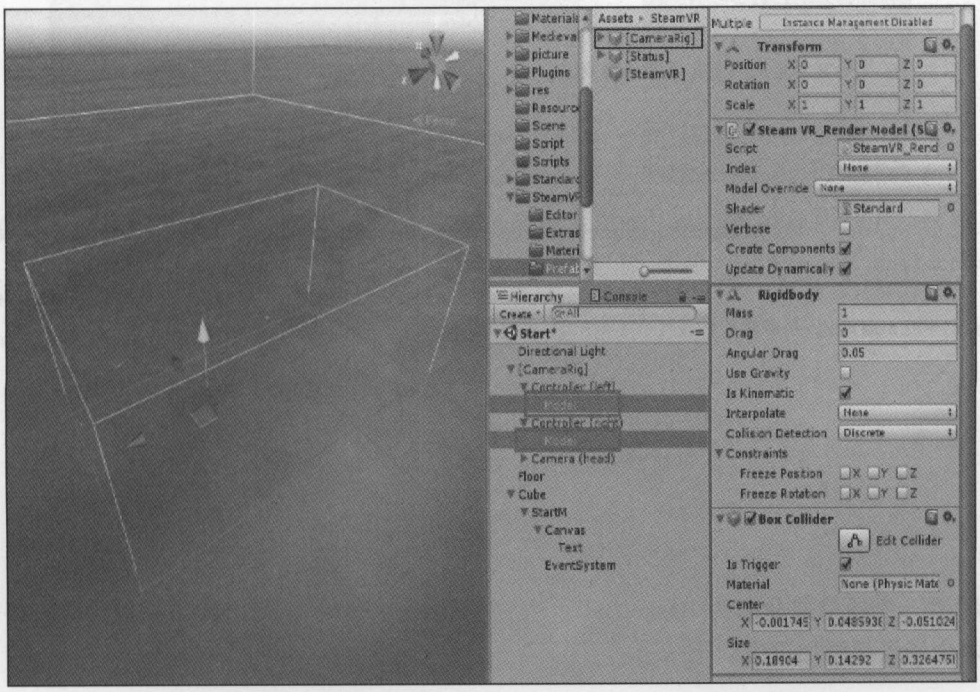


修改 Text 上边的文字为“开始”，并放到红色 Cube 的中间位置。

添加如下图所示的预制件并拖到场景中，该预制件包含 Vive 头盔显示器所适用的专用摄像机。

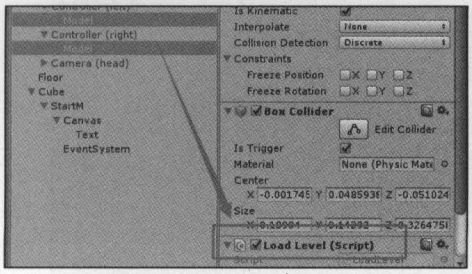

开始
picture
Plugins
res
Resource
Scene
Script
Scripts
Standard
Start_Cube1
Start_Cube2
Hierarchy
Console
Create
Start*
Directional Light
[CameraRig]
Floor
Cube
StartM
Canvas
Text
EventSystem
Rotation X 0 Y 0 Z 0
Scale X 0.5556 Y 1.7658 Z 1
Canvas Renderer
Text (Script)
Text
开 始
Character
Font Arial
Font Style Bold And Italic
Font Size 24
Line Spacing 1
Rich Text
Paragraph
Alignment
Align By Geometr
Horizontal Overflo Wrap
Vertical Overflow Truncate
Best Fit
Color
Material None (Material)
Raycast Target
Layout Properties

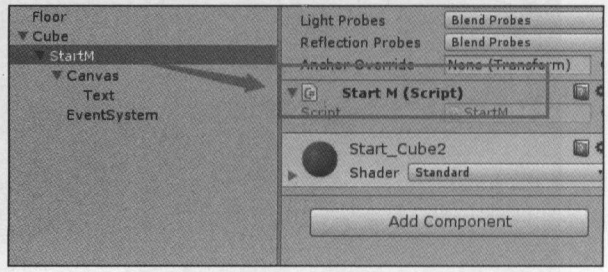

Project
Create
Materials
Medieval
picture
Plugins
res
Resource
Scene
Script
Scripts
Standard
SteamVR
Editor
Extras
Materi
Prefab
Assets ▸ SteamVR
[CameraRig]
[Status]
[SteamVR]


Assets ▸ SteamVR
[CameraRig]
[Status]
[SteamVR]
Hierarchy
Console
Start*
Directional Light
[CameraRig]
Controller (left)
Controller (right)
Camera (head)
Floor
Cube
StartM
Canvas
Text
EventSystem
Multiple
Instance Management Disabled
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Steam VR_Render Model (S
Script SteamVR_Rend
Index None
Model Override None
Shader Standard
Verbose
Create Components
Update Dynamically
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z
Box Collider
Edit Collider
Is Trigger
Material None (Physic Mate
Center
X -0.001745 Y 0.048593(Z -0.051024
Size
X 0.18904 Y 0.14292 Z 0.3264751

分别给左右手柄控制器添加刚体和碰撞器，并调节到合适的位置和大小。

创建两个 C# 脚本，分别命名为 StartM 和 LoadLevel，将 StartM 脚本拖到红色 Cube 上，将 LoadLevel 脚本拖到左右手柄模型上。





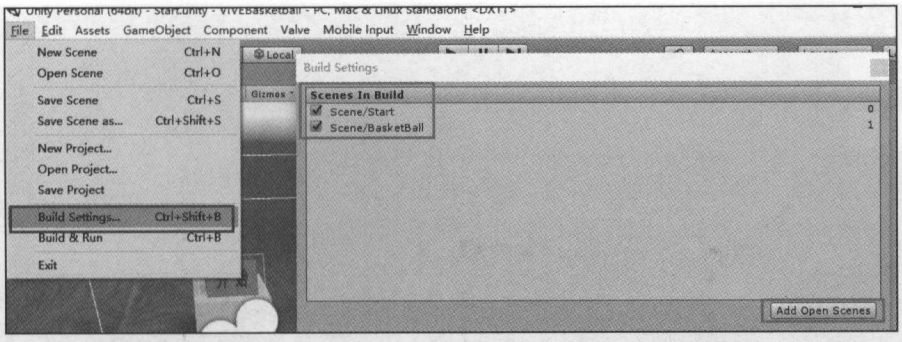
在 StartM 脚本中写入以下代码：

```
public class StartM : MonoBehaviour
    {
        public void OnButtonDown()
        {
            Application.LoadLevel(1);
        }
    }
```

在 LoadLevel 脚本中写入以下代码：

```
public class LoadLevel : MonoBehaviour
{
      public SteamVR_TrackedObject trackedobject;
      //获取当前的触发手柄
      void OnTriggerStay(Collider other)
      //使用触发器来判断是否开始游戏
      {
          StartM starbutton = other.GetComponent<StartM> ();
        if (starbutton != null)
                {
                  SteamVR_Controller.Device device=SteamVR_Controller.Input((int)
trackedobject.index);
                        if (device.GetTouchDown (SteamVR_Controller.ButtonMask.
Trigger)==true) //判断是否按下 Trigger（扳机键）
                  {
                        starbutton.OnButtonDown ();
                        //调用到红色 Cube 的开始按钮
                  }
            }
      }
}
```

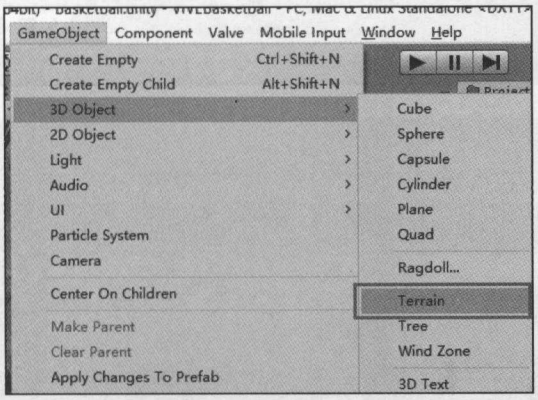
新建一个场景并命名为 BasketBall，然后按下 Ctrl+Shift+B 调出发布设置菜单，将创建好的两个场景都添加进去。

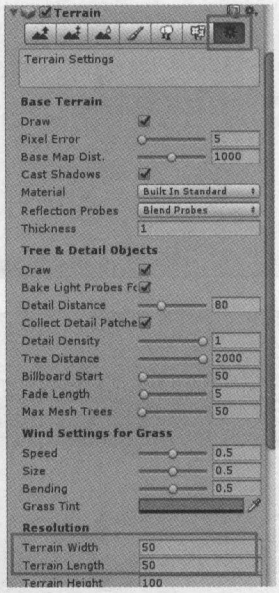


接下来点击运行进行测试，现在我们就可以从开始界面进入投篮游戏界面了。

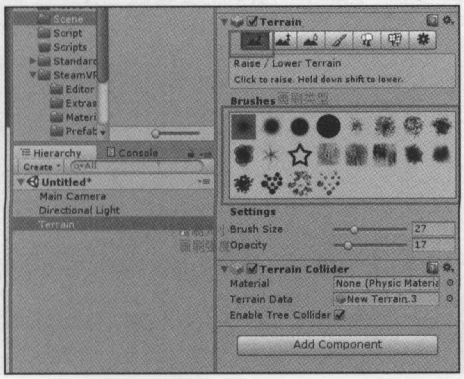
2. 投篮游戏的制作

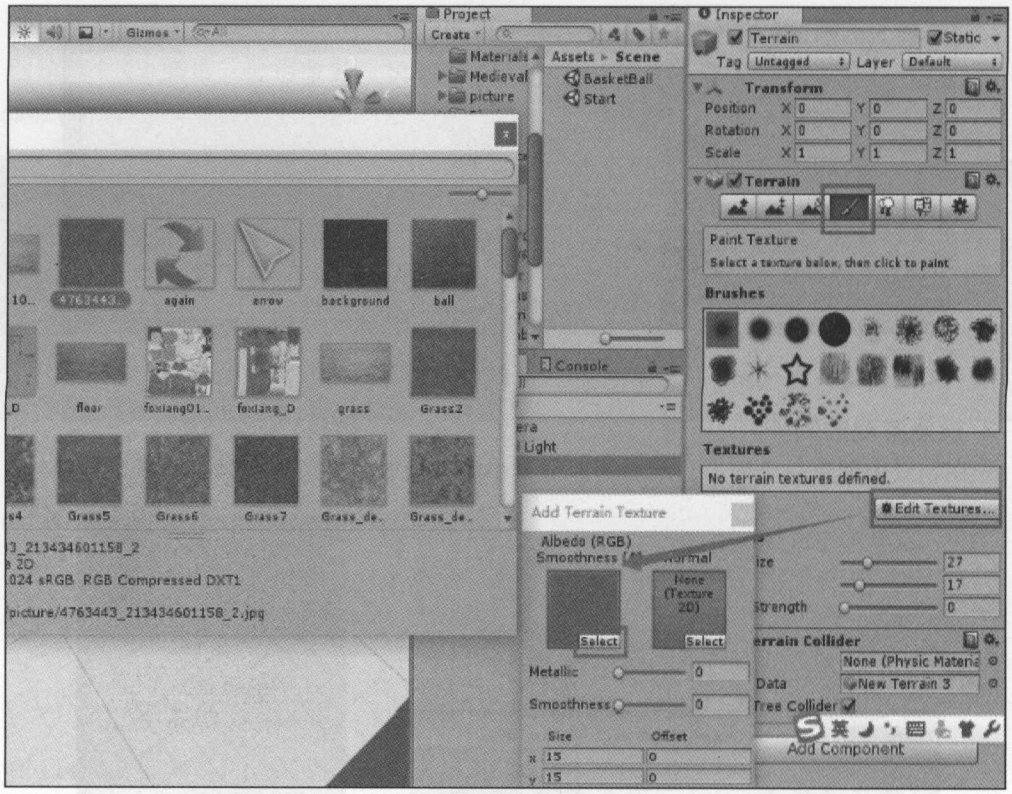
先在场景中创建一个地形，然后将地形大小调节为 50 × 50。



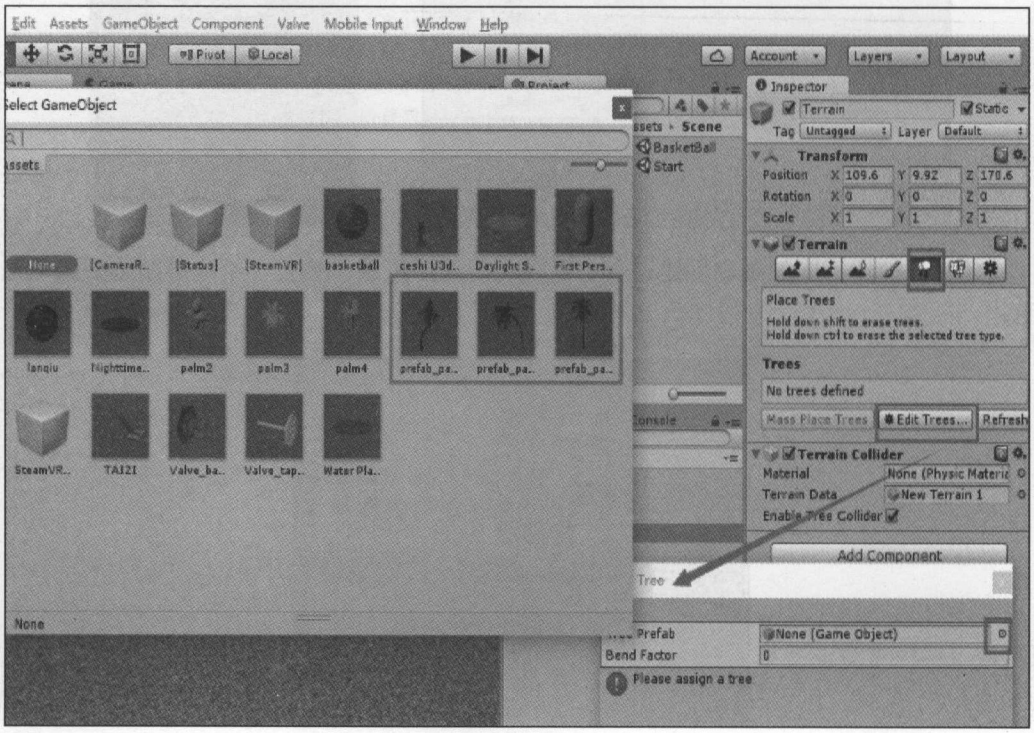


在此界面使用不同的画刷创建一个自己喜欢的地形风貌，然后再给地形贴上一张草地的贴图。





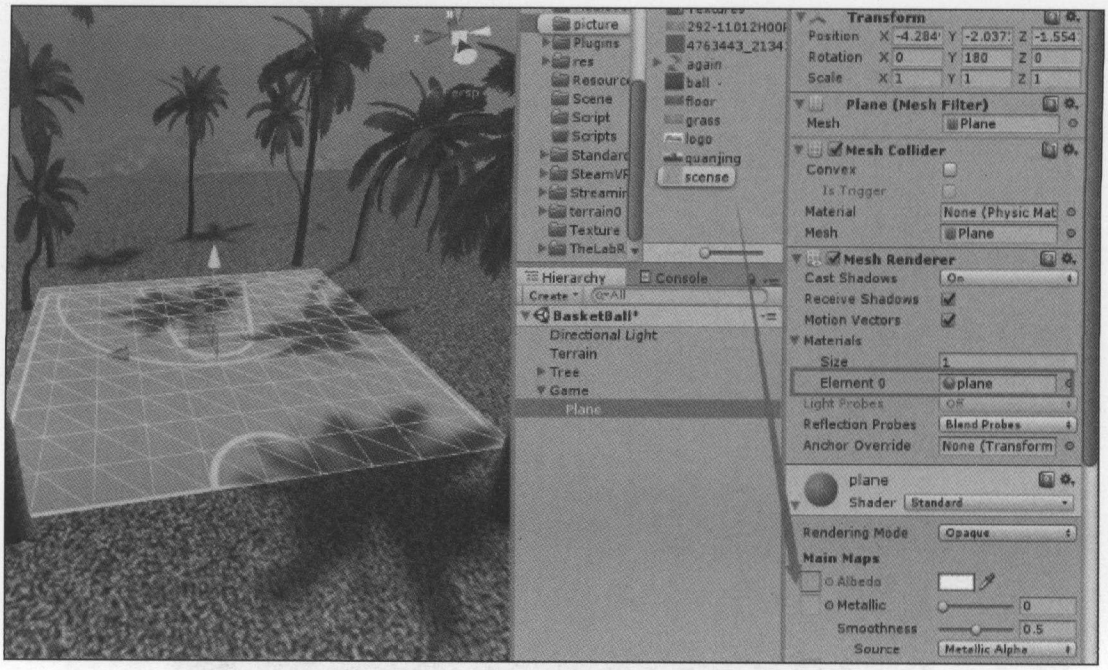
也可以在场景中添加一些树木，让整个场景更自然。





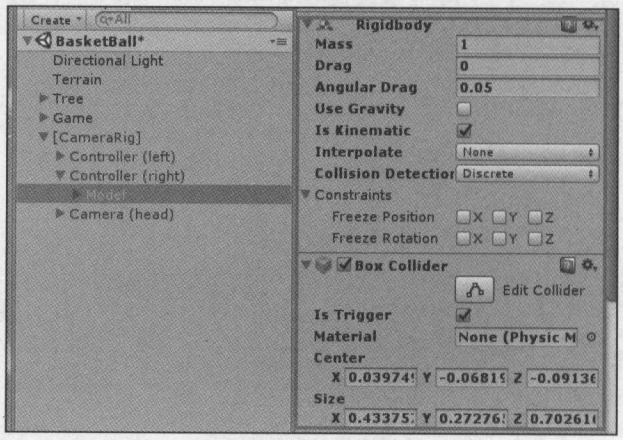
创建一个 Plane，将篮球地板贴图添加上去，并放到地形场景中。





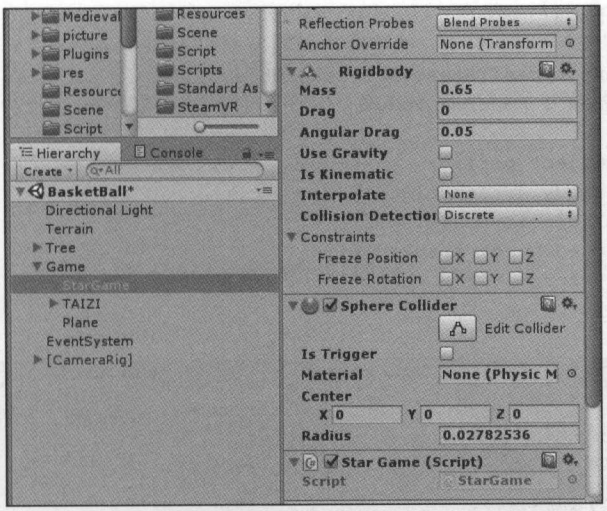
然后将在 3D Max 中制作好并导入进来的篮球架也拖到场景中。

将头显专用的摄像机拖到场景中，并放到球场中央的位置，同时给右手柄模型添加碰撞器和刚体。



将导进来的篮球拖到场景中作为游戏开始的触发器，同时创建一个 C# 脚本并命名为 StarGame，将其拖到篮球身上以使该篮球持续旋转。

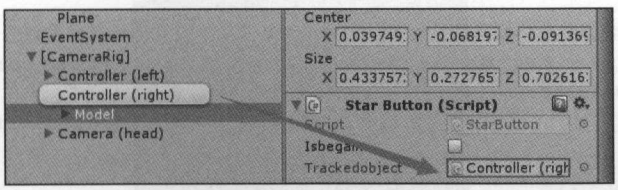
```
public class StarGame : MonoBehaviour
{
void Update ()
    {
            transform.Rotate (new Vector3 (0,0,3));
    }
}
```



当手柄碰到该篮球并扣动扳机之后投篮游戏正式开始，创建一个 C# 脚本并命名为 StarButton，将其拖到右手柄模型上，写入以下代码：

```
public class StarButton : MonoBehaviour
{
    public bool isbegain=false;// 判断游戏是否开始
    public SteamVR_TrackedObject trackedobject;
    void OnTriggerStay(Collider other)
    // 通过手柄与篮球的碰撞触发判断并开始游戏
    {
        StarGame star = other.GetComponent<StarGame> ();
        if (star != null)
        {
            SteamVR_Controller.Device device=SteamVR_Controller.Input((int)
trackedobject.index);
            if (device.GetTouchDown (SteamVR_Controller.ButtonMask.Trigger)==
true)

            {// 扣动手柄扳机键，游戏开始
                isbegain = true;
                Destroy (star.gameObject);
// 删除当前场景中的触发器，正式开始游戏
            }
        }
    }
}
```



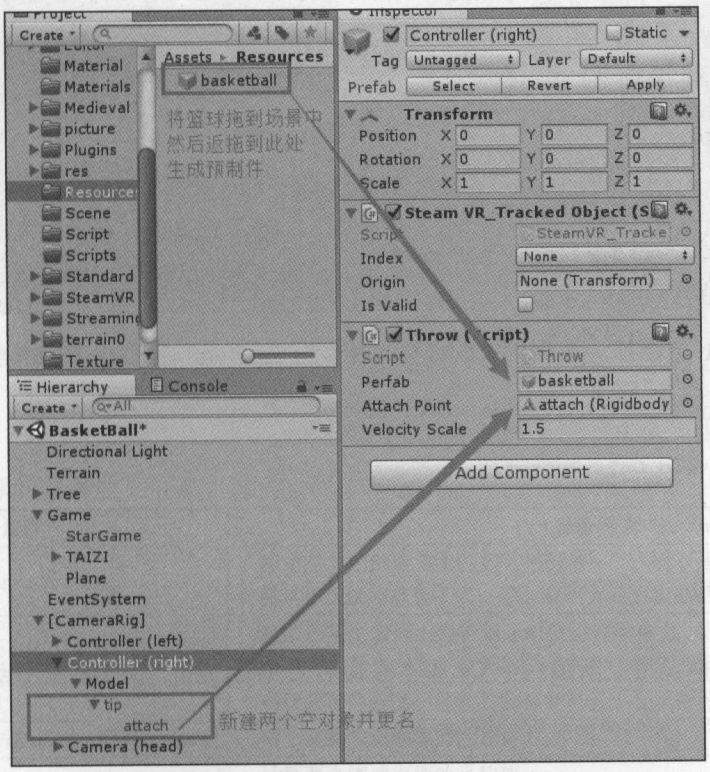
选择右手柄控制器，创建一个投篮的 C# 脚本并命名为 Throw，将其拖到右手柄上。双

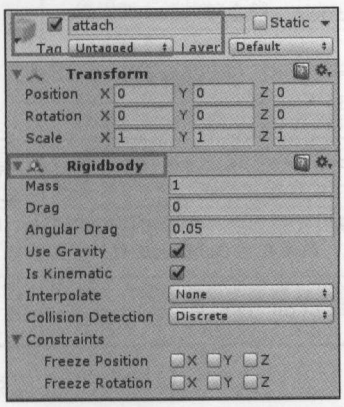
击打开该脚本进行编辑：

```
[RequireComponent(typeof(SteamVR_TrackedObject))]
public class Throw : MonoBehaviour
{
    public GameObject perfab;
    public Rigidbody attachPoint;
    public float VelocityScale = 3f;
    SteamVR_TrackedObject trackedObj;
    FixedJoint joint;
    void Awake()
    {
        trackedObj = GetComponent<SteamVR_TrackedObject>();
    }
    void FixedUpdate()
    {
        StarButton star = FindObjectOfType<StarButton> ();
        if (star.isbegain == true) // 判断游戏是否开始
        {
          var device = SteamVR_Controller.Input((int)trackedObj.index);
          if (joint == null && device.GetTouchDown(SteamVR_Controller.ButtonMask.
Trigger))// 检测到扳机键按下并且当前连接点为空的时候
          {
          // 通过预制件加载的方式实现加载篮球
          var go = GameObject.Instantiate (perfab);
          go.transform.position = attachPoint.transform.position;
          joint = go.AddComponent<FixedJoint> ();
          joint.connectedBody = attachPoint;
          }
            else if (joint != null && device.GetTouchUp(SteamVR_Controller.
ButtonMask.Trigger))
          // 检测到扳机键抬起同时物体的连接点不为空的时候
          {
          var go = joint.gameObject;
          // 获取当前抛出物体上的刚体
          var rigidbody = go.GetComponent<Rigidbody>();
          // 当物体被抛出时，删除 joint 组件
          Object.DestroyImmediate(joint);
          joint = null;// 规定 joint 为空，等待下一次物体生成
          Object.Destroy(go, 5.0f);
          Resources.UnloadUnusedAssets ();
          // 通过三目运算判断当前抛出对象是否有相对父对象的位置
             var origin = trackedObj.origin ? trackedObj.origin : trackedObj.
transform.parent;
          if (origin != null)
          {
          // 使当前物体刚体的速度等于相对父对象位置处扳机键松开时手柄的位移速度
              rigidbody.velocity = origin.TransformVector(device.
velocity * VelocityScale);
              rigidbody.angularVelocity = origin.TransformVector(device.
angularVelocity);
          }
          else
          {
```

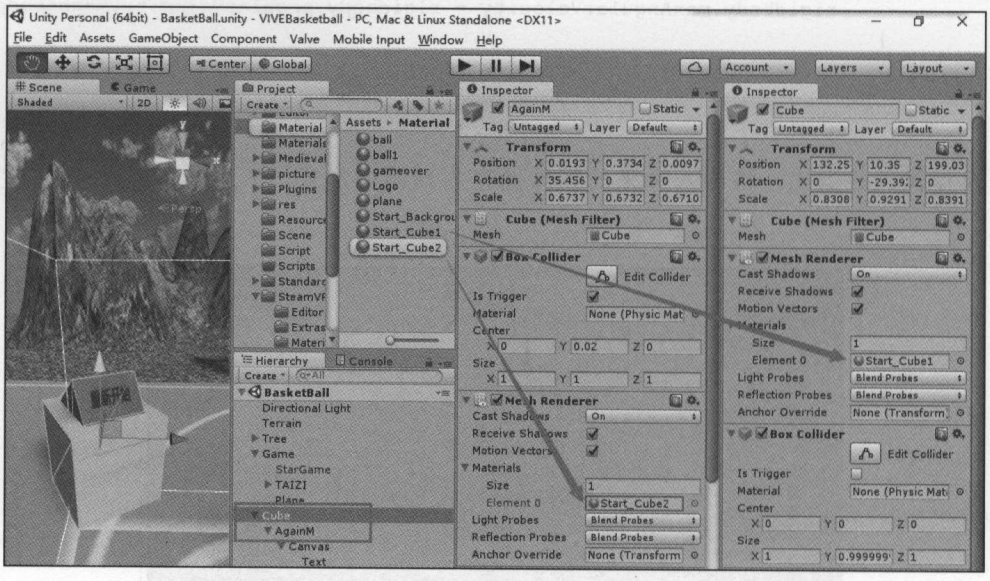
```
                rigidbody.velocity = device.velocity * VelocityScale;
                rigidbody.angularVelocity = device.angularVelocity;
            }
            rigidbody.maxAngularVelocity = rigidbody.angularVelocity.magnitude;
        }
        }
    }
}
```

保存代码并退出，将导入进来的篮球储存为预制件形式，然后选中右手柄并做如下配置。





在场景中新建两个 Cube，当作重新开始触发器，将开始场景中创建的材质球继续添加给该投篮场景中的 Cube，同时做如下配置。



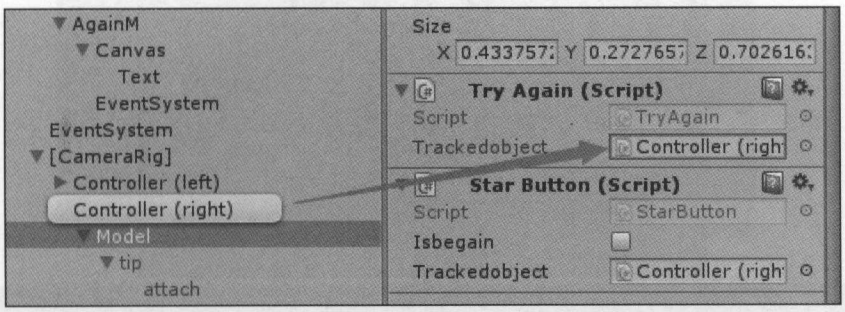
在红色 Cueb 上添加一个新的 C# 脚本，命名为 TryButton，双击打开并编辑如下代码：

```
public class TryButton : MonoBehaviour
{
    public void OnTryButton()
    {
        print ("重新开始");
        Application.LoadLevel (1);
    }
}
```

同时在右手柄模型上新建一个 C# 脚本并命名为 TryAgain，双击打开并编辑如下代码：

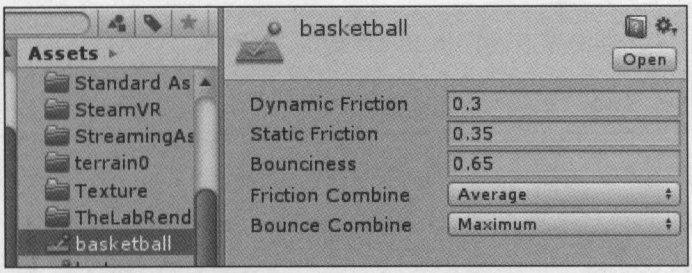
```
public class TryAgain : MonoBehaviour
{
  public SteamVR_TrackedObject trackedobject;
  void OnTriggerStay(Collider other)
  {
     TryButton trybutton = other.GetComponent<TryButton> ();
     if (trybutton != null)
     {
          SteamVR_Controller.Device device=SteamVR_Controller.Input((int)
trackedobject.index);
        if (device.GetTouchDown (SteamVR_Controller.ButtonMask.Trigger)==true)
        // 当扣动扳机键的时候，激活红色 Cube 上的重新开始按钮
        {
          trybutton.OnTryButton ();
        }
      }
   }
}
```

在右手柄模型上将右手柄控制对象托上去。



最后给篮球创建一个物理材质球，物理材质球可以用于调整碰撞对象的摩擦力和反弹效果，物理材质球的创建路径是 Assets → Create → Physics Material。

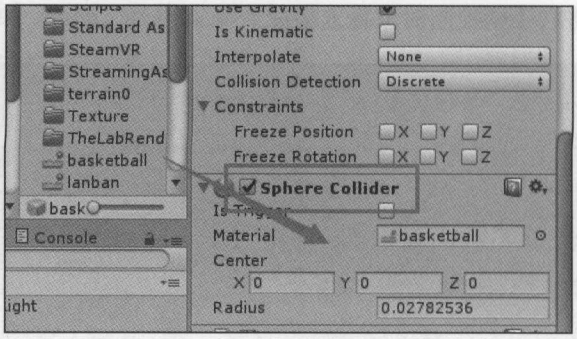
对创建好的物理材质球做如下参数调整：



物理材质球的各属性见下表：

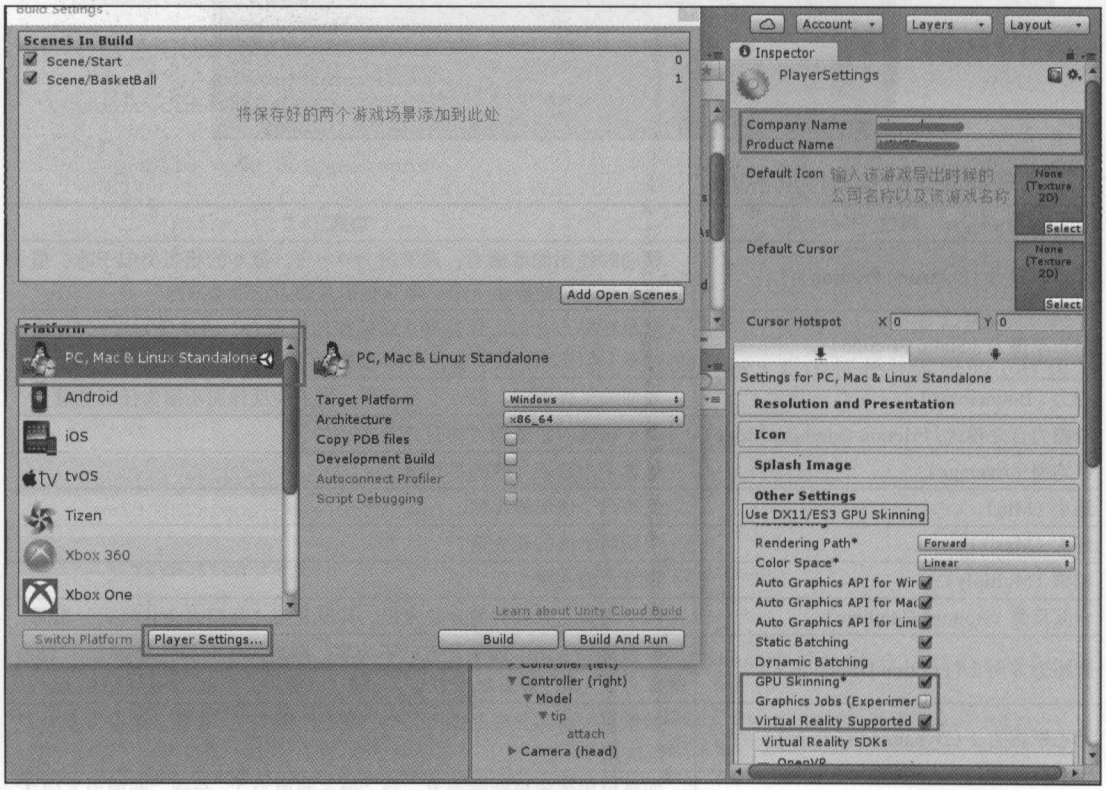
属性	功能
动态摩擦力（Dynamic Friction）	移动时使用的摩擦力，通常值为 0 ～ 1。值 0 的状态类似于冰，值 1 会使其非常快地静止下来，除非有很大的力在推动对象
静态摩擦力（Static Friction）	对象在某个表面上保持静止时使用的摩擦力，通常值为 0 ～ 1。值 0 的状态类似于冰，值 1 会使对象非常难以移动
弹力（Bounciness）	表面的反弹程度。值 0 不会反弹，值 1 会反弹而不损失任何能量
摩擦力合并模式（Friction Combine Mode）	两个碰撞对象摩擦力的合并方式
平均值（Average）	计算两个摩擦力值的平均值
最小（Min）	使用两个值的较小值
最大（Max）	使用两个值的较大值
相乘（Multiply）	摩擦力值相乘
合并反弹（Bounce Combine）	两个碰撞对象弹力的合并方式。其模式与“摩擦力合并模式”相同
摩擦力方向 2（Friction Direction 2）	各向异性方向。如果此方向不为零，则启用各向异性摩擦力。“动态摩擦力 2”和“静态摩擦力 2”会随“摩擦力方向 2”一起应用
动态摩擦力 2（Dynamic Friction 2）	如果启用各向异性摩擦力，则“动态摩擦力 2”会随“摩擦力方向 2”一起应用
静态摩擦力 2（Static Friction 2）	如果启用各向异性摩擦力，则“静态摩擦力 2”会随“摩擦力方向 2”一起应用

在创建好篮球的物理材质球之后把物理材质球拖到篮球身上即可。

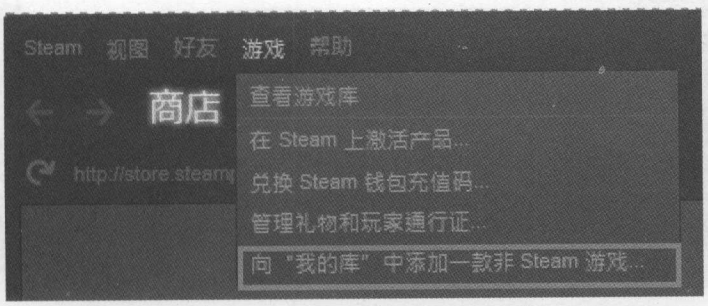


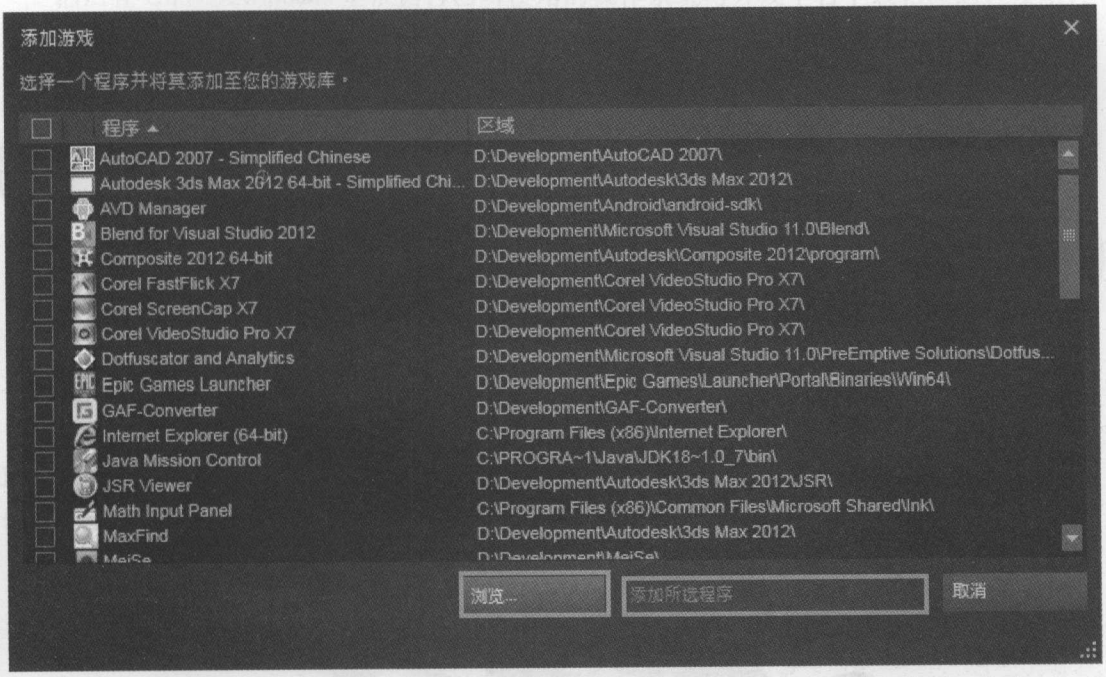
17.3 导出与运行

按下 Ctrl+Shift+B 调出导出设置，选择平台为 PC 平台，点击 Player Settings，做如下参数调整，最后点击导出（Build）。



将导出成功的 .exe 游戏文件添加到 Steam 游戏平台上。





最后就可以在 Steam 平台上运行投篮游戏了。

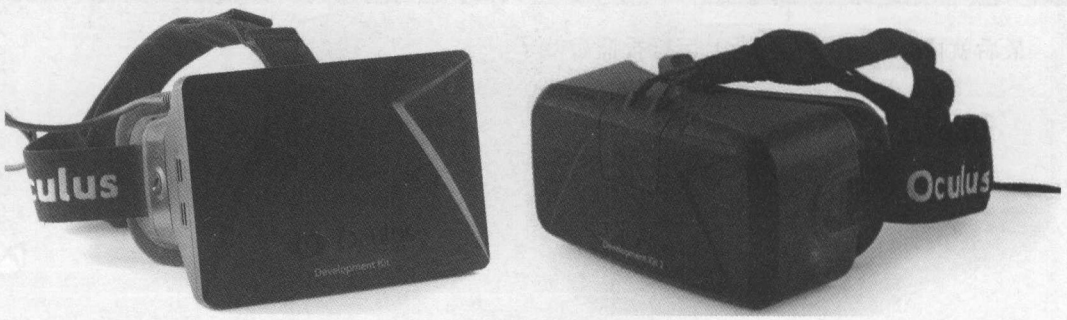
第18章 其他VR设备简介

18.1 Oculus VR简介

1. 设备介绍

Oculus成立于2012年，是目前世界上最大的VR品牌之一。在2014年7月的时候，Facebook以20亿美元的价格收购了Oculus，引起了社会各界的广泛关注。由此可见Facebook对于虚拟现实的态度，同时虚拟现实行业的未来也越来越被人们所关注。

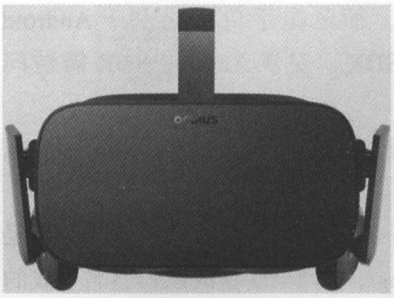
Oculus目前主要以硬件产品为主，Oculus Rift DK1和DK2（如下图）都属于开发者版本，Oculus Rift不只是一个硬件系统，而是包含软件开发工具包在内的一整套开发系统。用户使用该版本不仅可以体验到炫酷的VR，还可以开发自己的VR游戏。

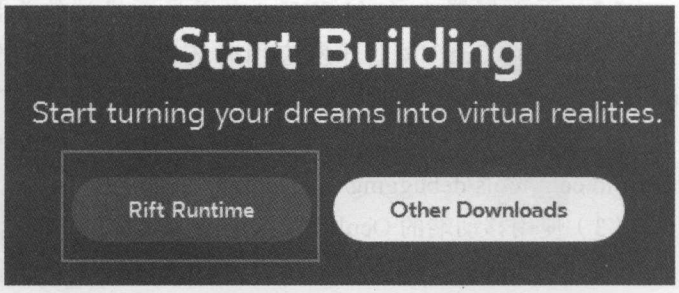


2016年1月7日零点，Oculus Rift CV1正式接受预订。这是Oculus Rift的零售版，包装内除了头盔外，还配备有一个Xbox手柄和两款自带游戏。

2. 开发流程

访问https://developer.oculus.com/注册自己的开发者账号并登录，下载Oculus Rift Runtime。





然后在 OtherDownloads 里根据自己所使用的操作系统与开发平台下载相应的开发组件与工具包。安装并调试好，就可以开发自己的 Oculus Rift 项目了。

18.2 GearVR 简介

1. 设备介绍

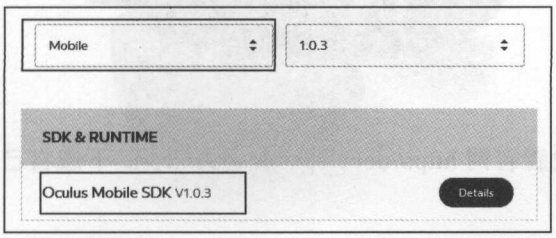
GearVR 是三星与 Oculus 合作推出的一款手机端 VR 头盔。消费者版本的 GearVR 在侧面添加了一个触摸板，用来完成各种各样的交互操作。无论是从设计还是体验来说，这都是一款非常优秀的 VR 头盔。

GearVR 的售价为 99 美元，由于是手机端的头盔，因此官方要求最低配置必须是三星 Note4 或者是 S6 才可以使用。如果你正好有这样一款手机，那么这个价格真的是无可挑剔，但是如果没有，那么成本无疑就高了。



2. 开发流程

（1）首先下载移动端 SDK，由于 GearVR 是与 Oculus 合作制作的，所以开发部分还是基于 Oculus 移动端的 SDK，依然是在 https://developer.oculus.com/ 创建开发者账号并下载移动端 SDK。



（2）下面配置 Android SDK，由于是手机端的头显，而三星手机又是基于 Android 平台，所以在 GearVR 上开发应用时还需要 Android 的 SDK。安装 Android SDK 需访问 developer.android.com/sdk/installing/index.html，

安装完成后，有可能需要进行调试，下面是一个很好的安卓调试工具：developer.android.com/tools/debugging/index.html。

（3）使用移动端的 Oculus SDK 构建的 VR 应用需要一个唯一的签名才可以调用更底层的 API。访问 Oculus 的开发者官网 developer.oculus.com/osig/，关于 ID 获取以及签名生成的具体方法都有详细的说明。

（4）想要在手机上构建部署自己的应用，需要先打开 USB 调试，关于这些，可访问 http://forums.androidcentral.com/samsung-galaxy-tab-pro-8-4/371491-enabling-developer-mode.html。

至此，环境搭建完成，开始你的 GearVR 应用开发吧。

18.3 大朋 VR 简介

1. 设备介绍

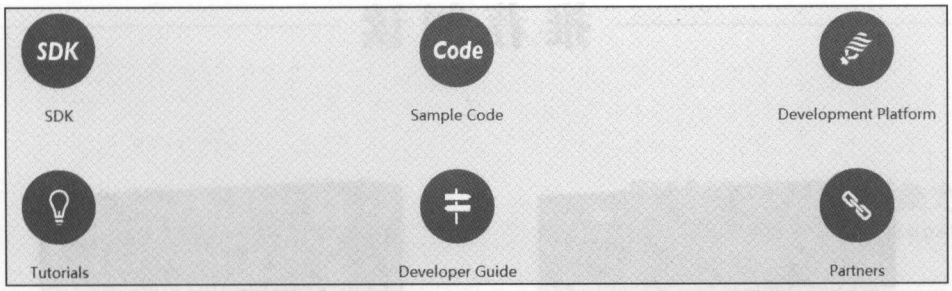
大朋 VR 是目前国内领先的虚拟现实互联网企业，拥有国际水平的 PC 端和移动端虚拟现实头戴产品，目前国内市场占有率达到 68%。它还拥有最大的 VR 内容聚合平台 3D 播播，平台用户数高达 300 万。大朋 VR 由国内第一个可穿戴计算机实验室的核心人员和对虚拟现实技术有狂热爱好的梦想家所建立，坚持以用户体验为中心，做最受用户喜爱的可穿戴设备，公司员工多是来自 Intel、ARM、百度、腾讯、盛大和 PPTV 等公司的顶尖高手，另有可穿戴计算、虚拟现实、增强现实、人工智能领域的世界知名科学家顾问团队。

大朋 VR 专注于虚拟现实头戴显示产品的开发和虚拟现实内容平台的建设，致力于打造以硬件系统为核心、软件内容同步发展、聚合游戏及影视等多方资源的 VR 生态。

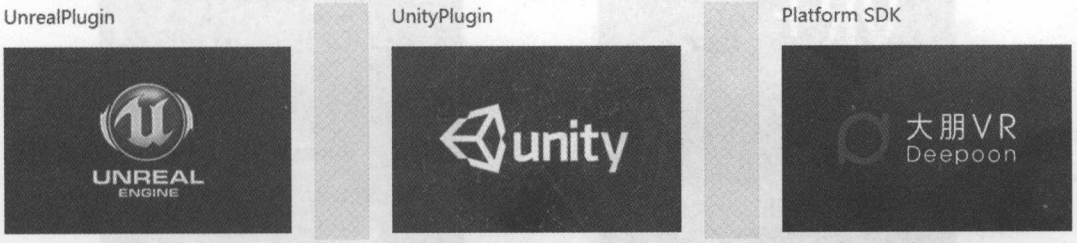


2. 开发流程

访问大朋 VR 开发者官网 http://developer.deepoon.com/ 下载自己所需要的 SDK 或者其他的资源包。



目前支持以下三种平台：

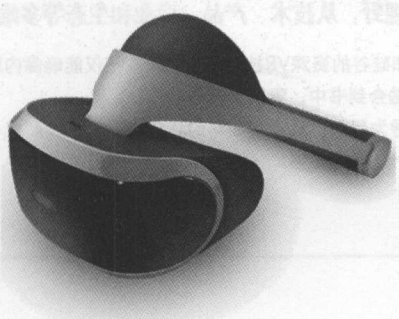


SDK 及官网中都有非常详细的文档介绍说明，大家下载安装后就可以在相应的平台上进行开发了。

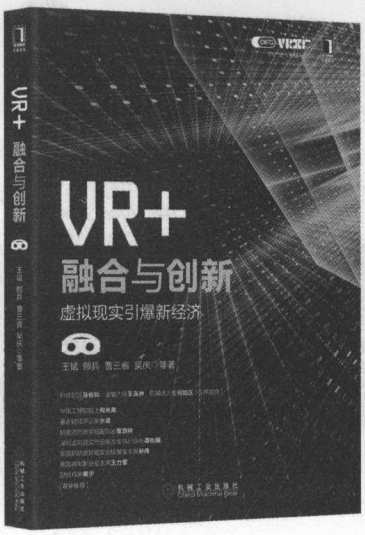
18.4 PSVR 简介

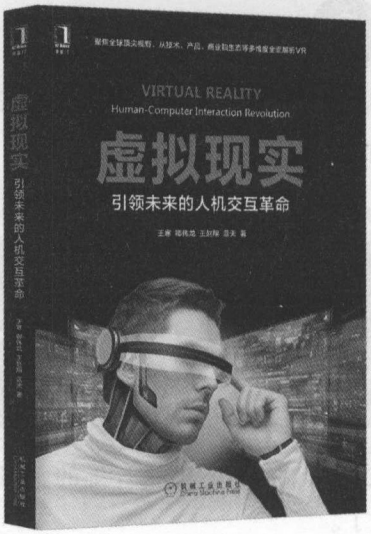
索尼公司在游戏圈的地位毋庸置疑，VR 游戏如此火爆，这位行业大佬怎能无动于衷？2016 年 3 月 16 日，索尼正式推出 PSVR，立刻引起了广泛关注，当天下午，亚马逊上的第一批设备在短短几分钟内售罄。

PSVR 的售价为 399 美元，但这只是头盔的价格，全套 PSVR 设备还包括一台 PS4 主机和 PSVR 摄像头等，一共需要 869 美元。这个价格相对于 Oculus 和 HTC Vive 来说是一个优势，更何况 PSVR 不需要额外再配一台高配置的主机。所以对于想要体验高级别 VR 设备的读者来说，PSVR 无疑是最具诱惑的，再加上索尼在游戏方面多年的经验积累，PSVR 值得期待。



推荐阅读





VR+：融合与创新

作者：王斌 等 ISBN：978-7-111-54799-0 定价：49.00元

系统阐述“VR＋”创新体系的经典读本，看虚拟现实如何引爆新经济

深度揭示“VR＋产业”的商业模式以及投资商机，揭秘VR与电影、游戏、旅游、教育、房地产、汽车、媒体、电商、医疗等传统行业融合创新所产生的巨变与发展

vr+：2016年是VR产业元年，VR产业进入高速增长期，未来十年将成为超过万亿的产业。虚拟现实VR技术在影视、游戏、消费、旅游、教育、房产、医疗、体育等各个领域的创新融合，将为产业带来巨变！本书梳理了国内外VR产业发展现状与趋势、VR商业模式以及我国VR产业版图及VR产业投资。将重点以VR+产业为核心，首次揭示VR与电影、游戏、旅游、教育、房地产、汽车、媒体、电商、医疗等行业融合创新所产生的巨变与发展。

虚拟现实：引领未来的人机交互革命

作者：王寒 等 ISBN：978-7-111-54111-0 定价：59.00元

聚焦全球顶尖视野，从技术、产品、商业和生态等多维度全面解析VR

作者团队由来自国内和硅谷的资深VR技术专家组成，不仅能确保内容的专业性，而且能把国内和国外的VR行业状况全部融合到书中，更加全面和立体。

本书以科普和商业引导为目的，从现实、科幻、技术、产品、商业、生态6个维度对VR进行全方位的呈现，了解VR，这一本书就足够！

THINK 加速想象力
Thinker Boost

加速想象力为广大AR/VR爱好者准备了与本书配套的免费在线课程，可以通过网站http://edu.arinchina.com/寻找适合自己的课程。

为了更好地帮助读者根据本书进行学习，也可扫描下方二维码关注我们的订阅号“加速想象力”以了解更多资讯。

封面设计 · 王建敏



张明军 ARinChina CEO

任何行业的发展都需要坚实的人才基础，ARinChina能够在行业早期以社区及媒体为平台，以行业人才培养和服务切入行业生态，相信借此将会推动国内AR/VR产业成熟走出更坚实的一步。



廖春元 亮风台CEO

AR/VR是下一代计算平台，将广泛应用于各行各业，发展前景巨大。相关研发人才的培养是整个产业链中关键的一环，对行业的持续健康发展和从业人员的职业规划有非常重要的意义。本书的出版弥补了这个空缺环节，是AR/VR行业蓬勃发展的明证，值得推荐！



赵刚 京东VR实验室技术负责人

PC是桌面上的互联网，智能手机是口袋中的互联网，而AR/VR则是眼镜里的互联网，这又是一场已经近在咫尺的互联网颠覆式革新，游戏、影视、社交、电商将悉数被重新定义，让我们拥抱革新吧，本书是革新者的有力武器。



陈东义 电子科技大学教授

AR/VR技术是当今前沿与热点的技术领域，将创造新的市场，也将给传统行业带来变革。技术变革是推动人类进步的阶梯，人才是发展的根本动力。未来将需要更多的AR/VR开发人才，本书的推出非常及时，适合当前对人才培养的迫切需求。融合虚拟与现实，无界共享时空！



祖厚超 青瞳视觉CEO

AR/VR作为下一代数字体验平台、下一块屏幕，需要大量的行业开发者，本书很好地契合了市场的需求。



武博文 RAVV亚洲太平洋地区秘书长

AR/VR带来的人机交互的新革命将人类与数字信息完美结合，创造力、想象力突破了屏幕的限制且被最大化释放。最早掌握核心技术的造梦者们，前途不可限量。



翁孝清 上海微景文化传媒有限公司CEO

虚拟现实是未来世界的入口，本书是为开拓未来的人提供地图。

上架指导：AR/VR开发

ISBN 978-7-111-55330-4

9 787111 553304 >

定价：69.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn